ICTP: Wireless Sensor Networks Workshop

Instructors: Rob Faludi & Jordan Husney

Plan

- Introductions
- Radio
- XBees
- Serial Terminals
- Addressing
- Basic Config
- Chat Project
- I/O Mode
- Doorbell Project
- ZigBee
- Arduino & XBee
- API
- Sensor Networks
- Gateways
- XIG, iDigi, Dia
- Workshop, Q&A

Instructor Introductions

- Who we are
- What we do
- Most important thing we would teach you, (if we could!)

Student Introductions

- Name, where you are from, what you do
- Experience with electronics and programming: new, some, lots
- What you want out of these workshops
- Desired superpower

[Fun with XBees Presentation]

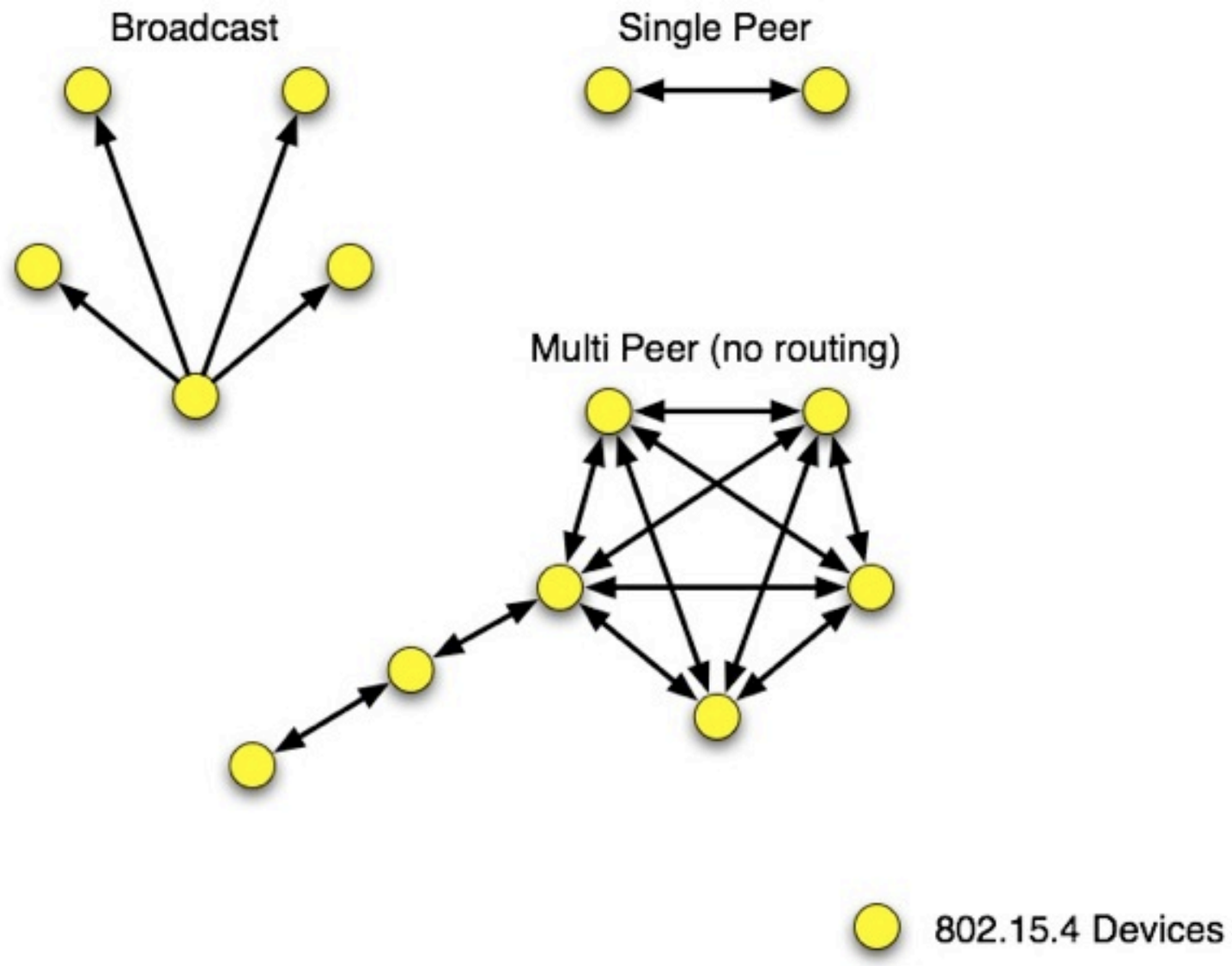
[Industrial Applications of WSN]

802.15.4

- low power
- low bandwidth
- addressing
- affordable
- small
- standardized
- popular for DIY, easy to learn

802.15.4 Topologies

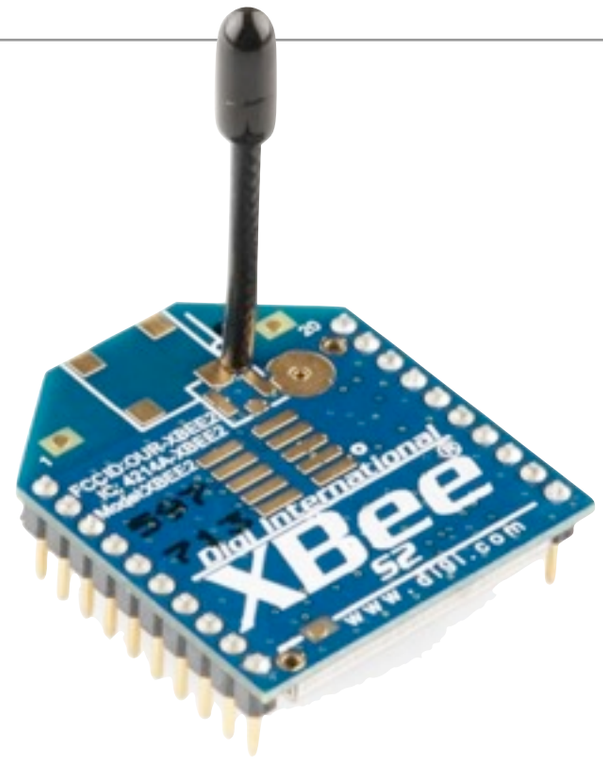
- single peer
- multi-peer
- broadcast



ZigBee

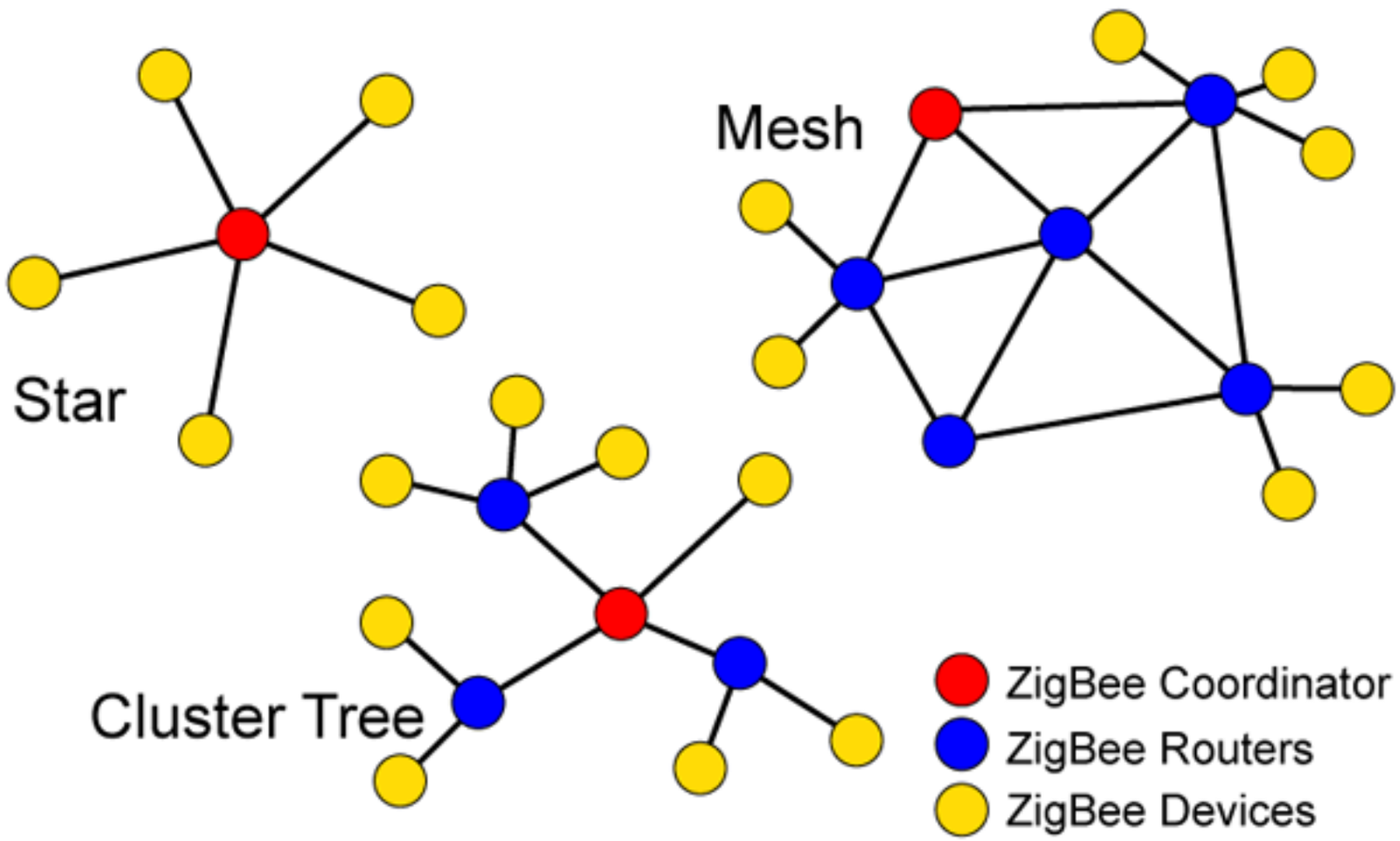
- routing
- self-healing mesh
- ad-hoc network creation

ZigBee™ Alliance

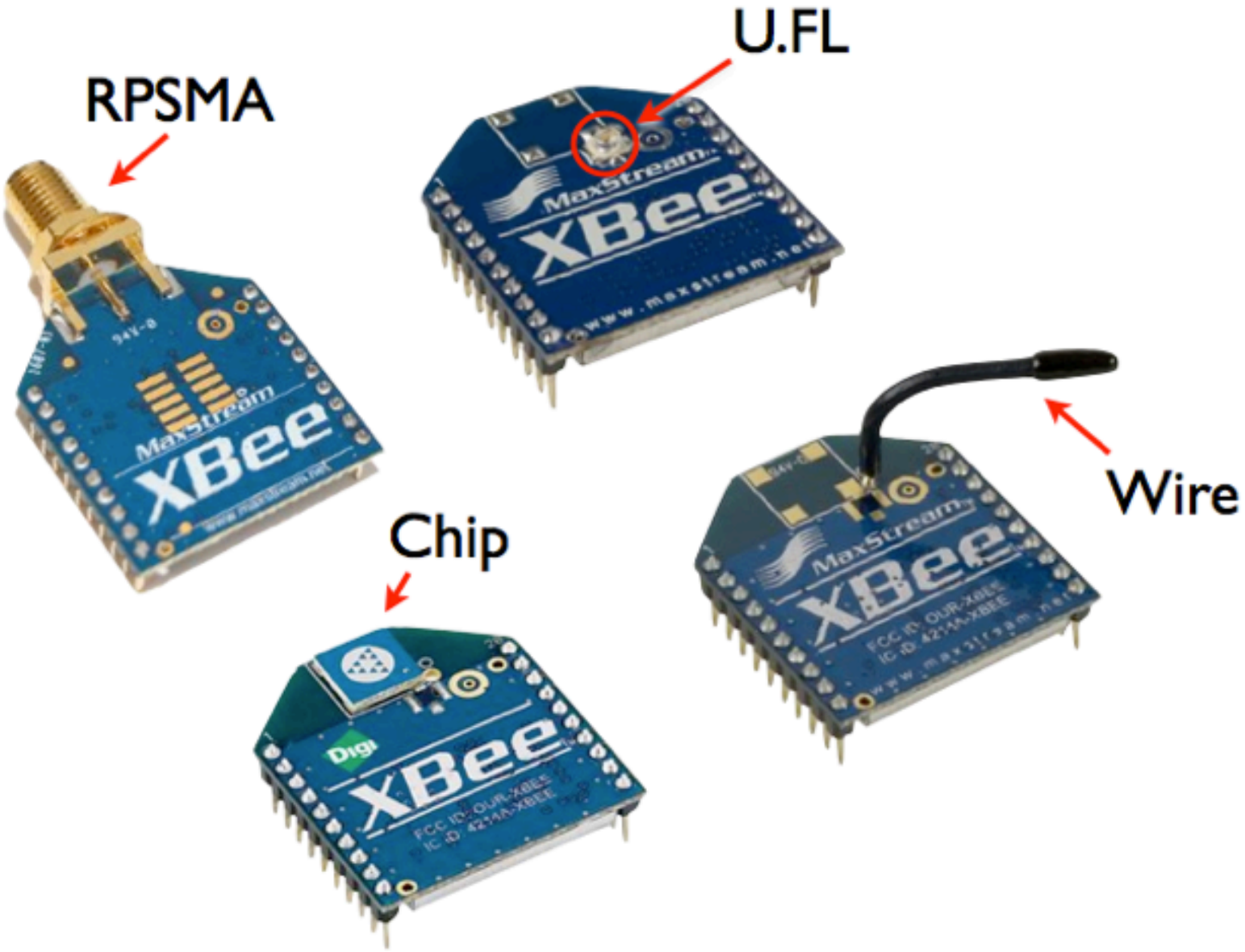


ZigBee Topologies

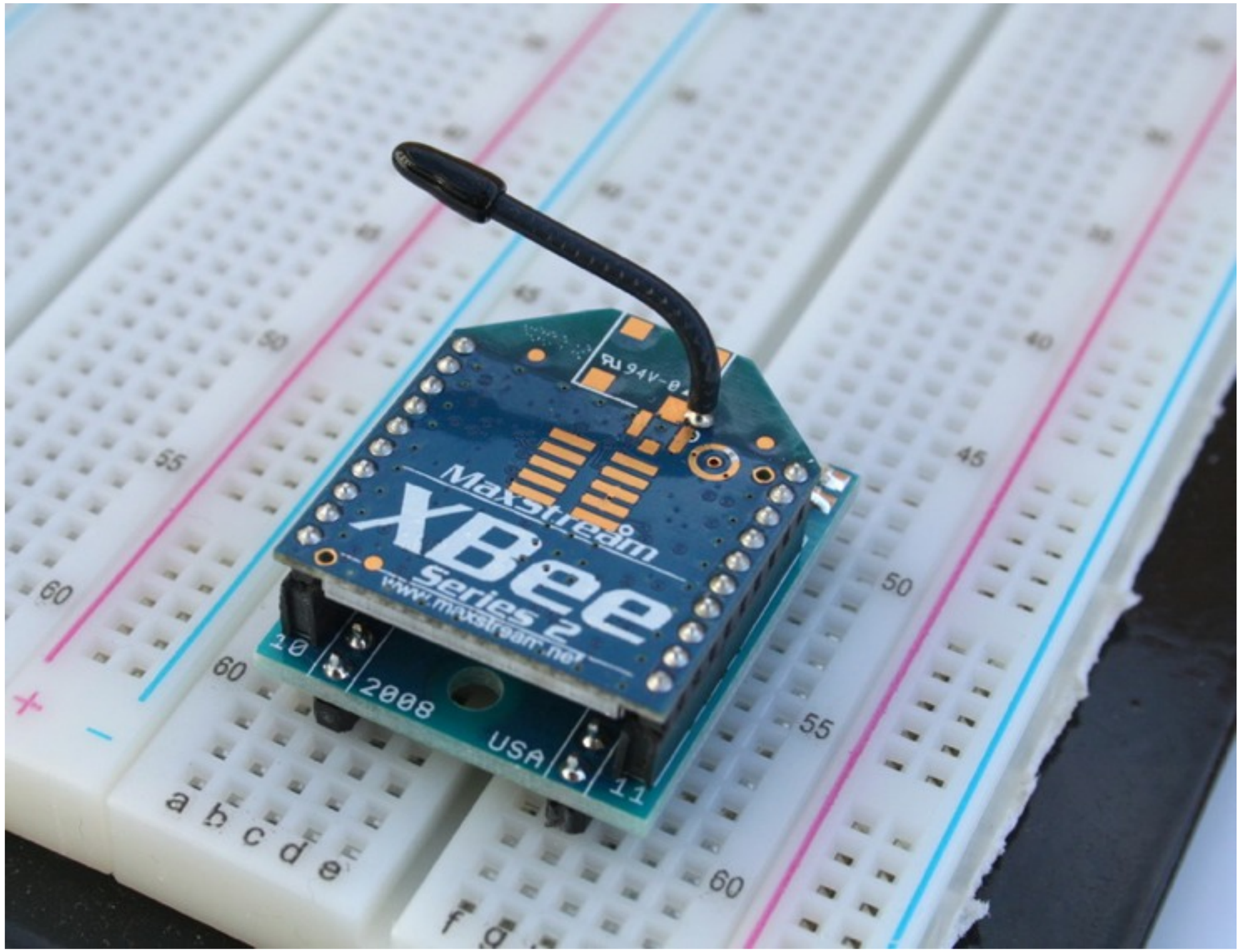
- peer
- star
- mesh
- routing



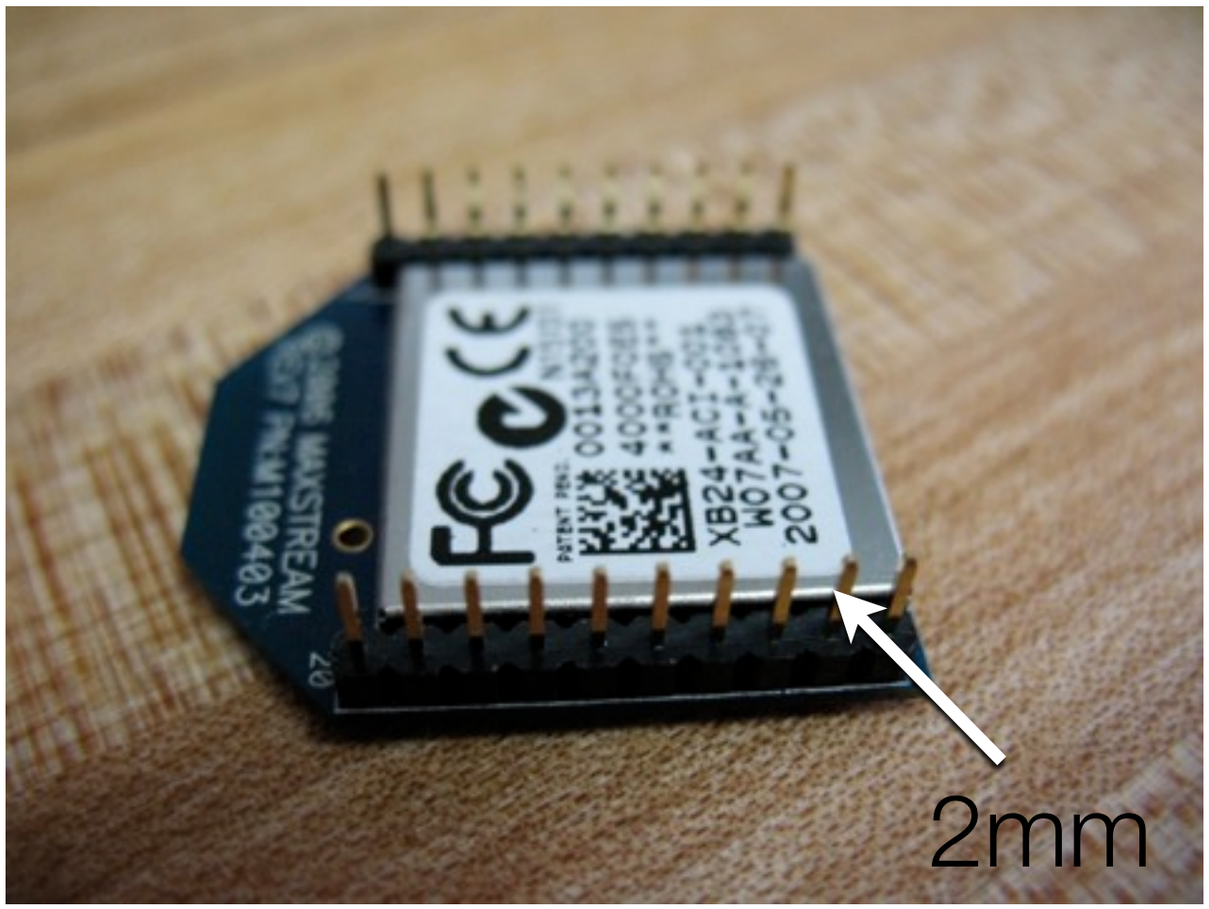
Antennas

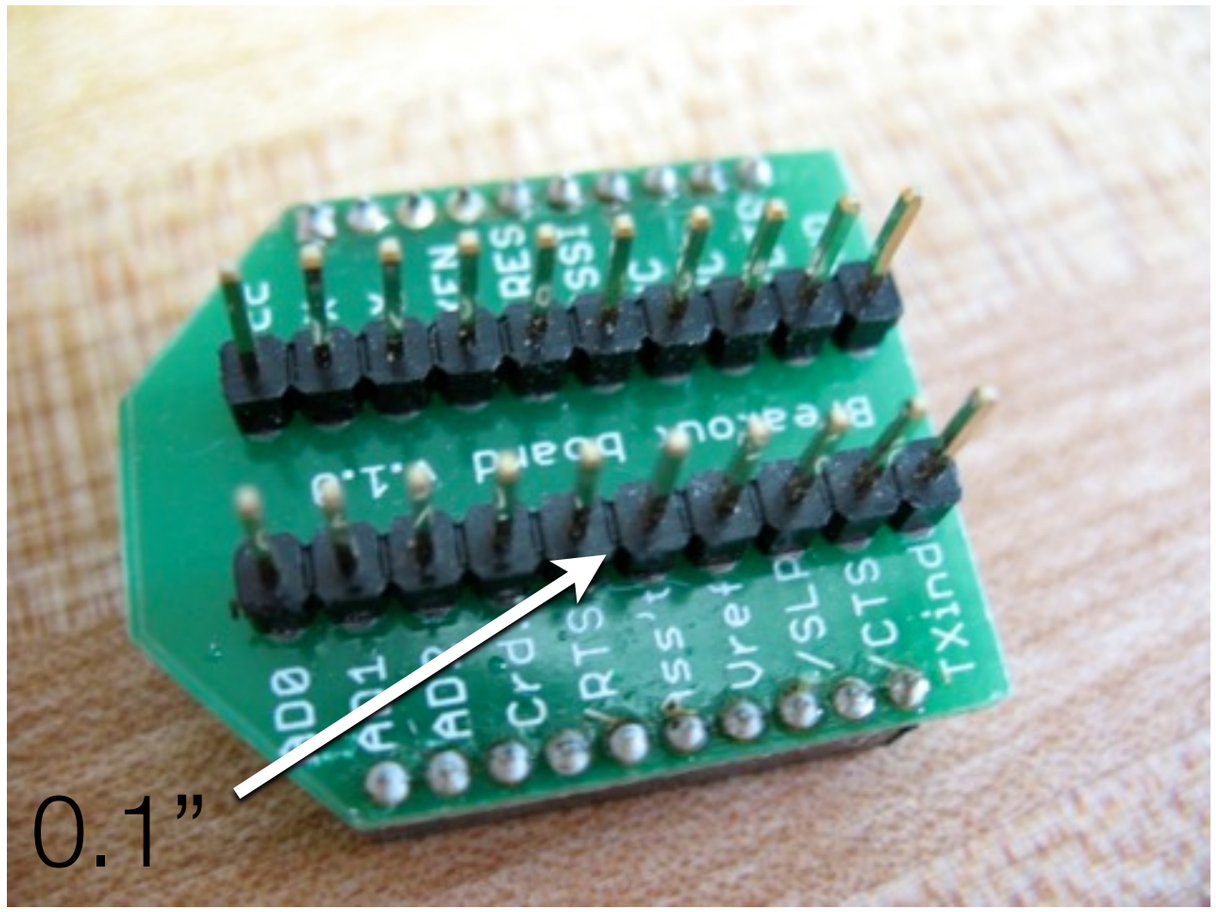


Breakout for Breadboards

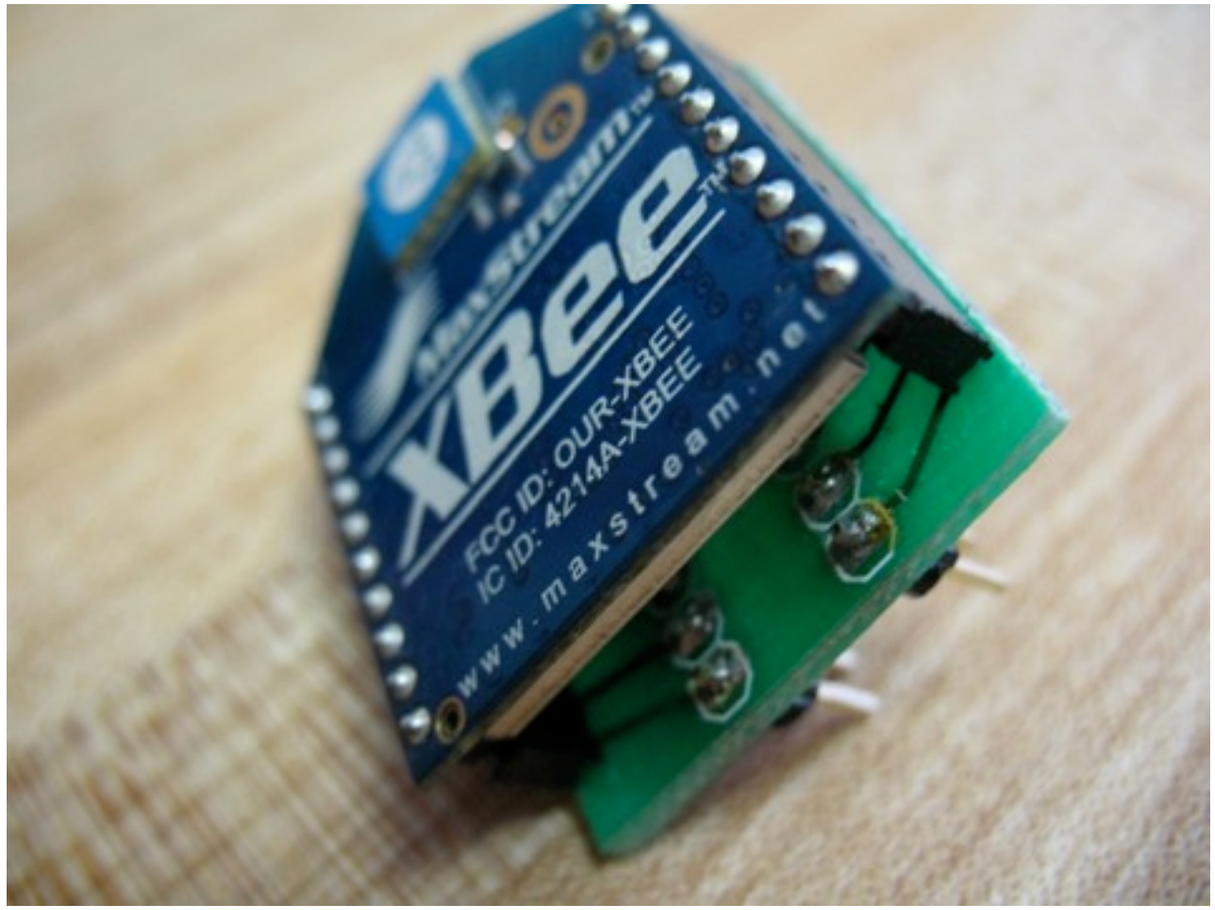


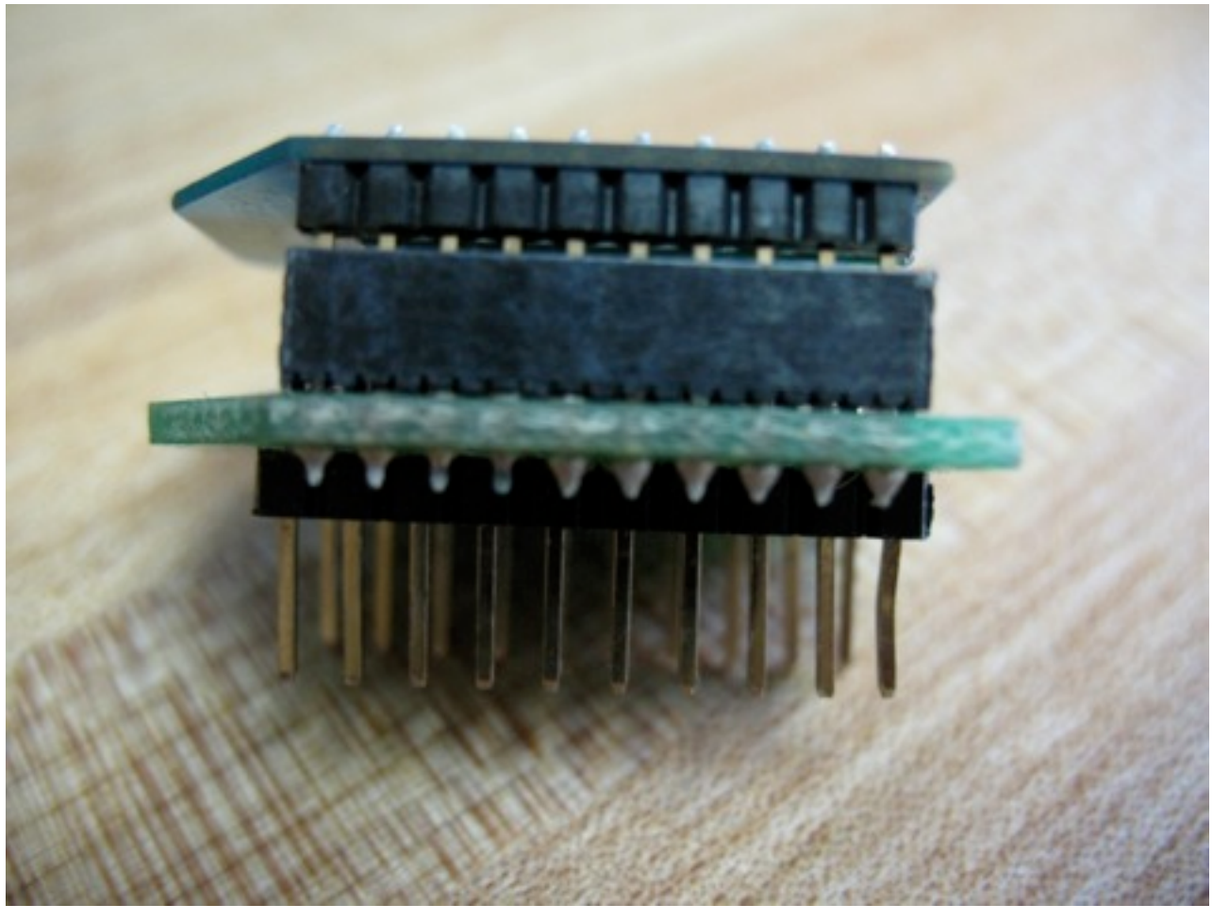
Breakout Boards for breadboarding



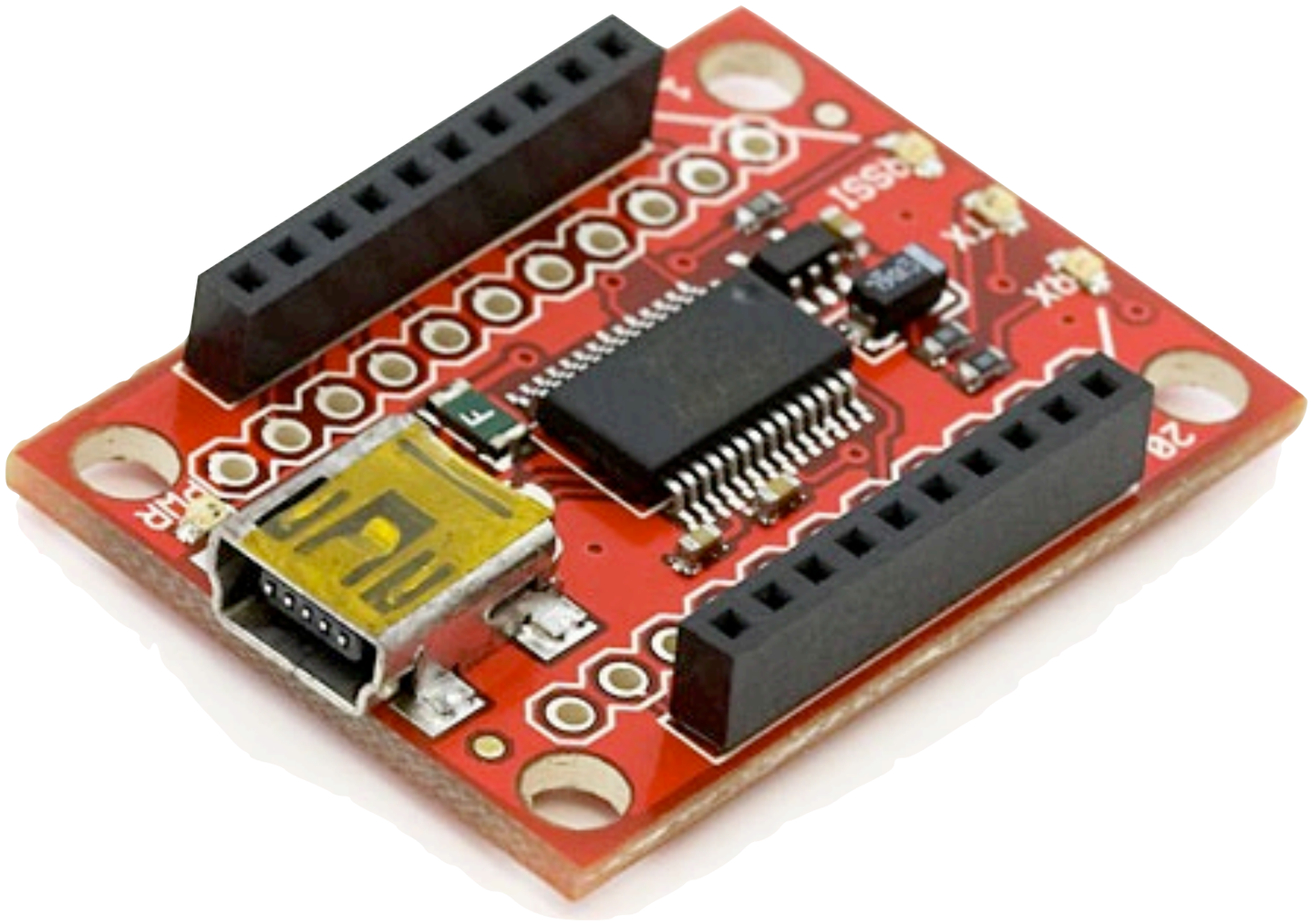


Soldering Breakout Boards: finished

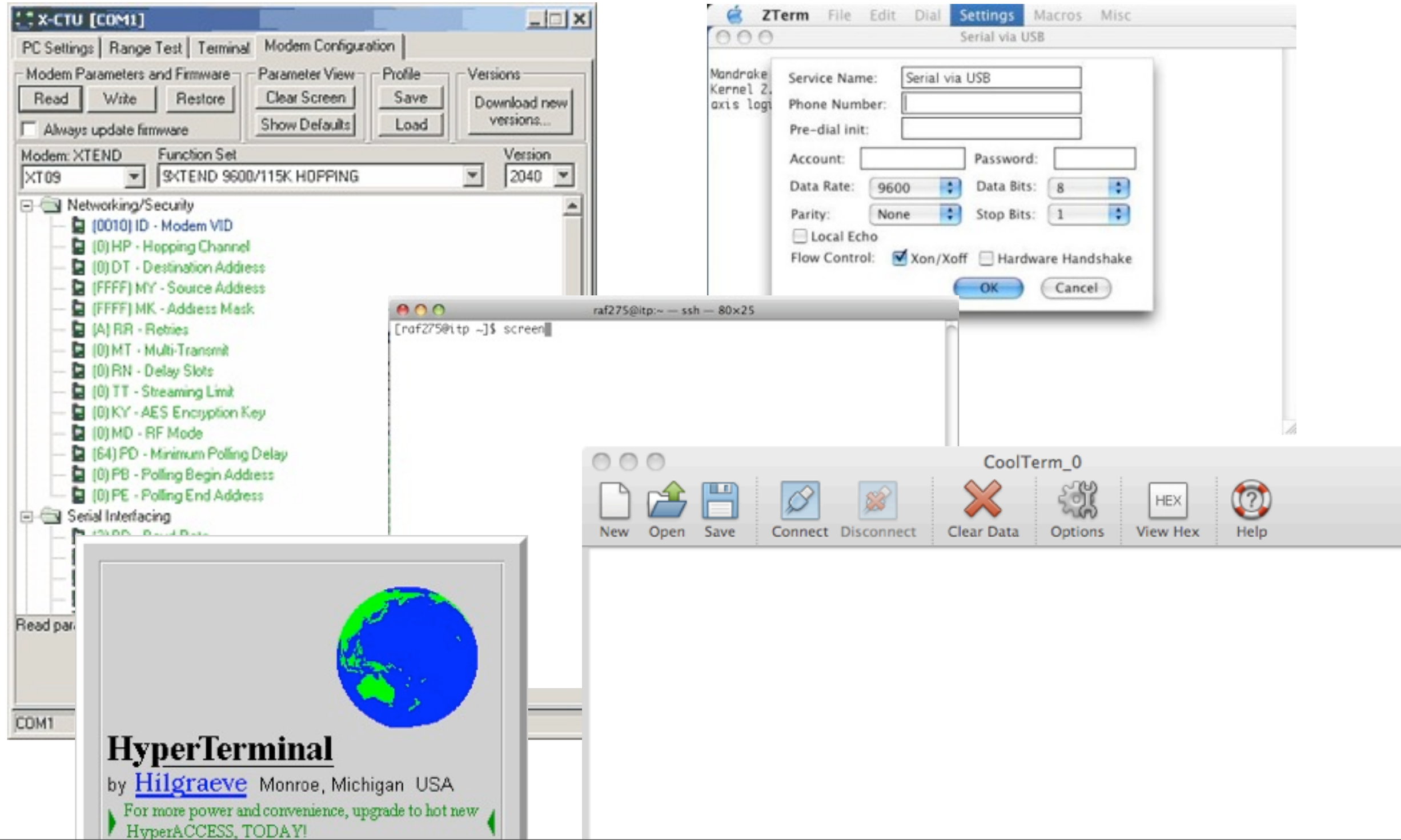




XBee Explorer from Sparkfun



Serial Terminal Programs



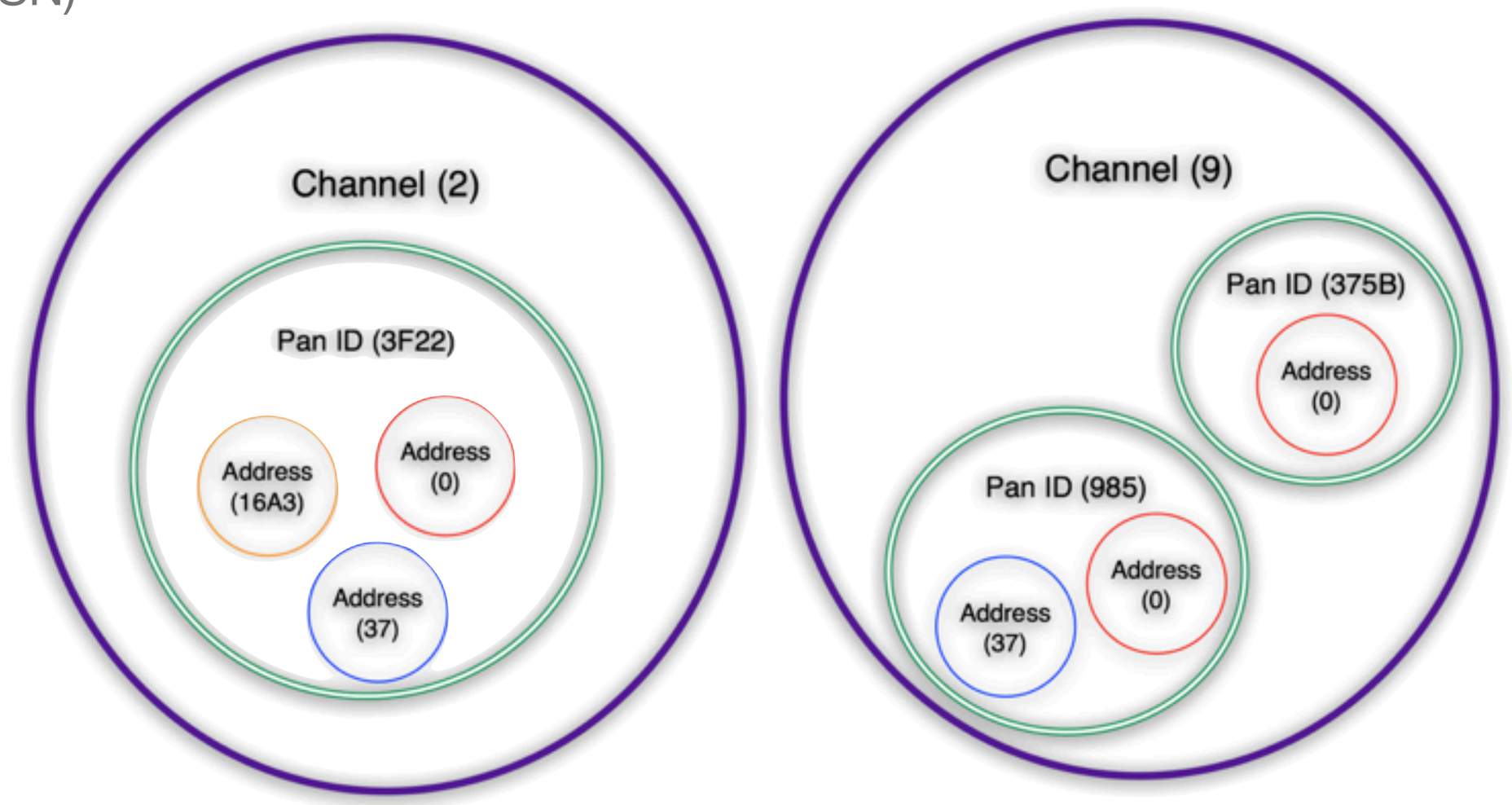
Serial Terminal Programs

- X-CTU: http://www.digi.com/support/productdetl.jsp?pid=3352&osvid=57&tp=4&s=316
- CoolTerm: http://freeware.the-meiers.org/
- HyperTerm: Windows Start Menu, Accessories, Communication http://www.hilgraeve.com/hyperterminal/
- screen: Terminal program on the Mac (or Linux)
- plenty of others!
- settings: 9600 baud, 8 bits, no parity, one stop bit, no flow control

802.15.4 Addressing

Addressing Basics

- channels
- PAN ID
- 64 bit addresses (SN)
- 16 bit addresses



Basic Configuration

Download and Install Software & Drivers

- Download & install the FTDI USB drivers:
  http://www.ftdichip.com/Drivers/VCP.htm

- Download the CoolTerm:
  http://freeware.the-meiers.org/

Other Serial Terminal Options:
settings: 9600 baud, 8 bits, no parity, one stop bit, no flow control

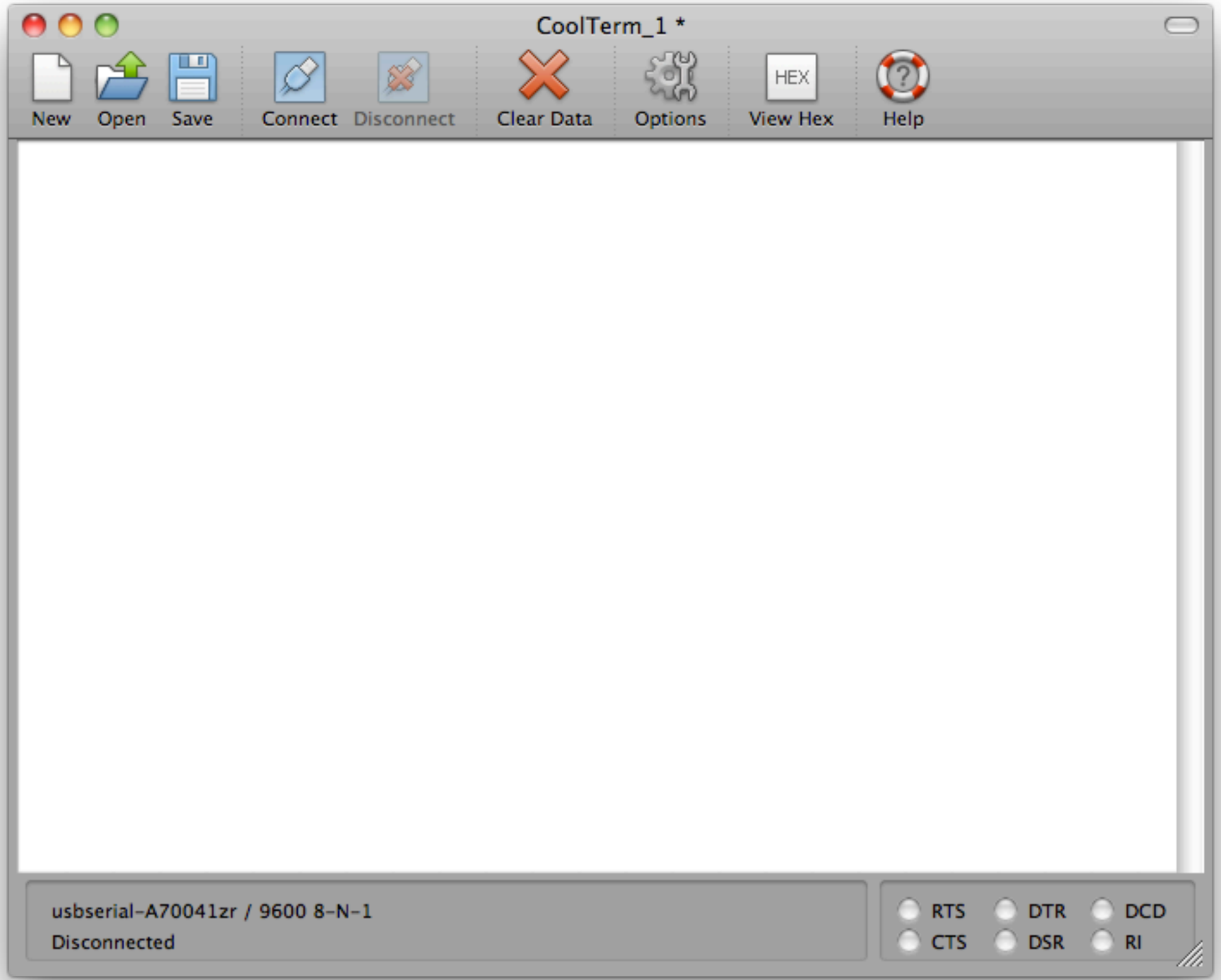
- X-CTU: http://www.digi.com/support/productdetl.jsp?pid=3352&osvid=57&tp=4&s=316

- Z-Term: http://homepage.mac.com/dalverson/zterm/

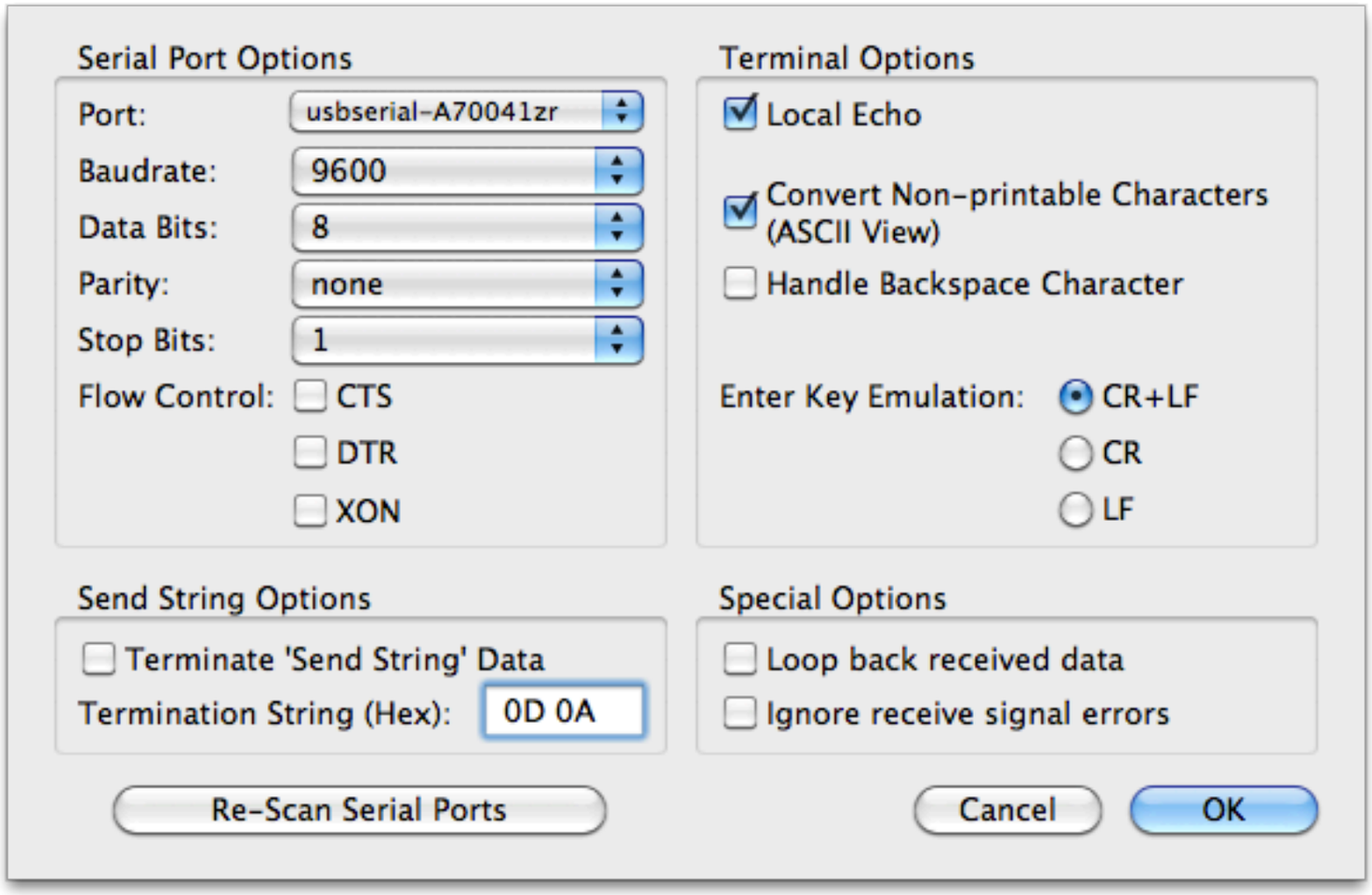
- HyperTerm: Windows Start Menu, Accessories, Communication

- Screen: Terminal program on the Mac (or Linux)

Open CoolTerm

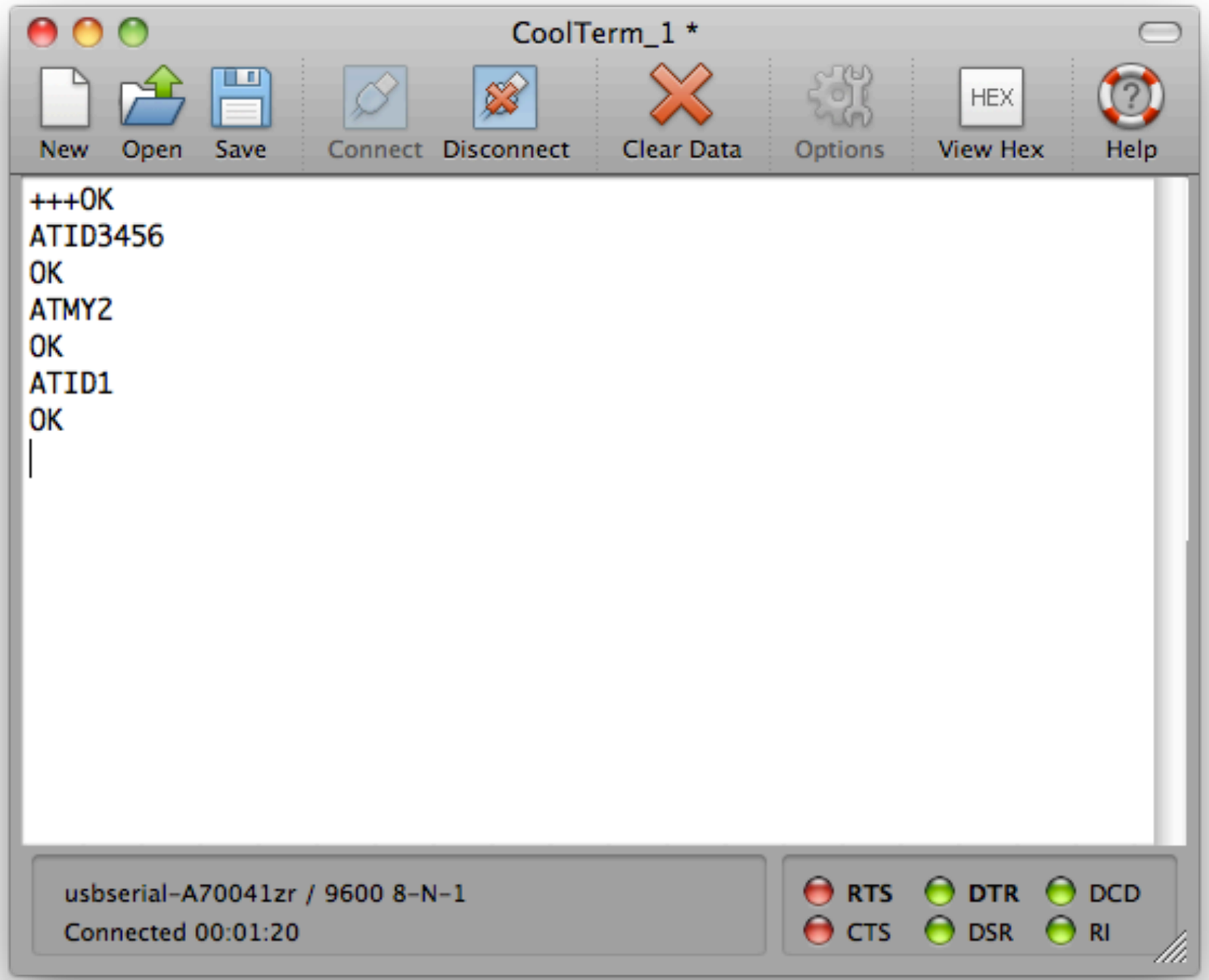


Set Connection Options



Configure your radio with AT commands

- Configure your radio



Baud, Bits and Parity

- Baud rate: 9600
- Data bits: 8
- Stop bits: 1
- Parity: None
- Flow control: none for now...

Data Mode vs. Command Mode

- Idle Mode, transmit and receive data
- Command Mode, talk to the XBee itself
  - +++ "*Yo, XBee*"
  - AT "*Attention!*" (Hayes command set)
    - always press enter after AT commands
    - *never* press enter after +++

AT Commands

Some AT Commands

- AT -> OK
- ATMY -> my address
- ATDH, ATDL -> destination address hi/lo
- ATID -> personal area network ID
- ATCN -> end command mode
- ATWR -> write configuration to flash memory
- ATRE -> reset to factory defaults

Addressing In-Depth

- SL, SH: fixed serial number address
- MY: configured local 16 bit address
- DH, DL: destination address low and high
- ID: Personal Area Network ID
- Broadcast FFFF
- Broadcast PAN FFFF

API Mode

- Powerful, steeper learning curve
- Data wrapped together with commands, addressing and status information

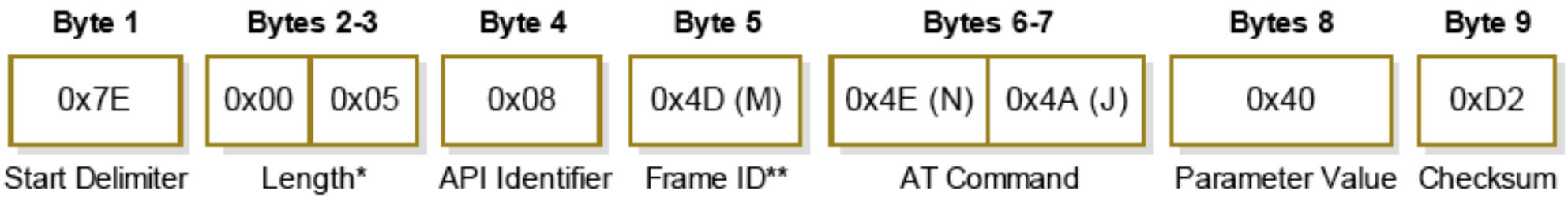
API Mode Format

Figure 4-01. UART Data Frame Structure:

MSB = Most Significant Byte, LSB = Least Significant Byte

Any data received prior to the start delimiter is silently discarded. If the frame is not received correctly or if the checksum fails, the module will reply with a module status frame indicating the nature of the failure.

Figure 4-07. Example: API frames when modifying the NJ parameter value of the module.



** Length [Bytes] = API Identifier + Frame ID + AT Command + Parameter Value*

*** "M" value was arbitrarily selected.*

*ATNJ = node join

Assignment

- Pick a PAN ID now and document it.
  - 0 - FFFE
  - 0 - 9999 okay

Basic 802.15.4 Chat

Create a Basic 802.15.4 Pair

- Two radios
- Use the 16-bit addresses for destinations
- HANDOUT
- Remember, the radios work reliably, troubleshooting is mostly about figuring out what they’re doing.

Ding, Dong!

Basic Doorbell

XBee Direct: no external microcontrollers:

1. doorbell switch connected to an XBee radio

2. buzzer connected to another XBee radio sounds the alert

3. someone’s at the door!

Background

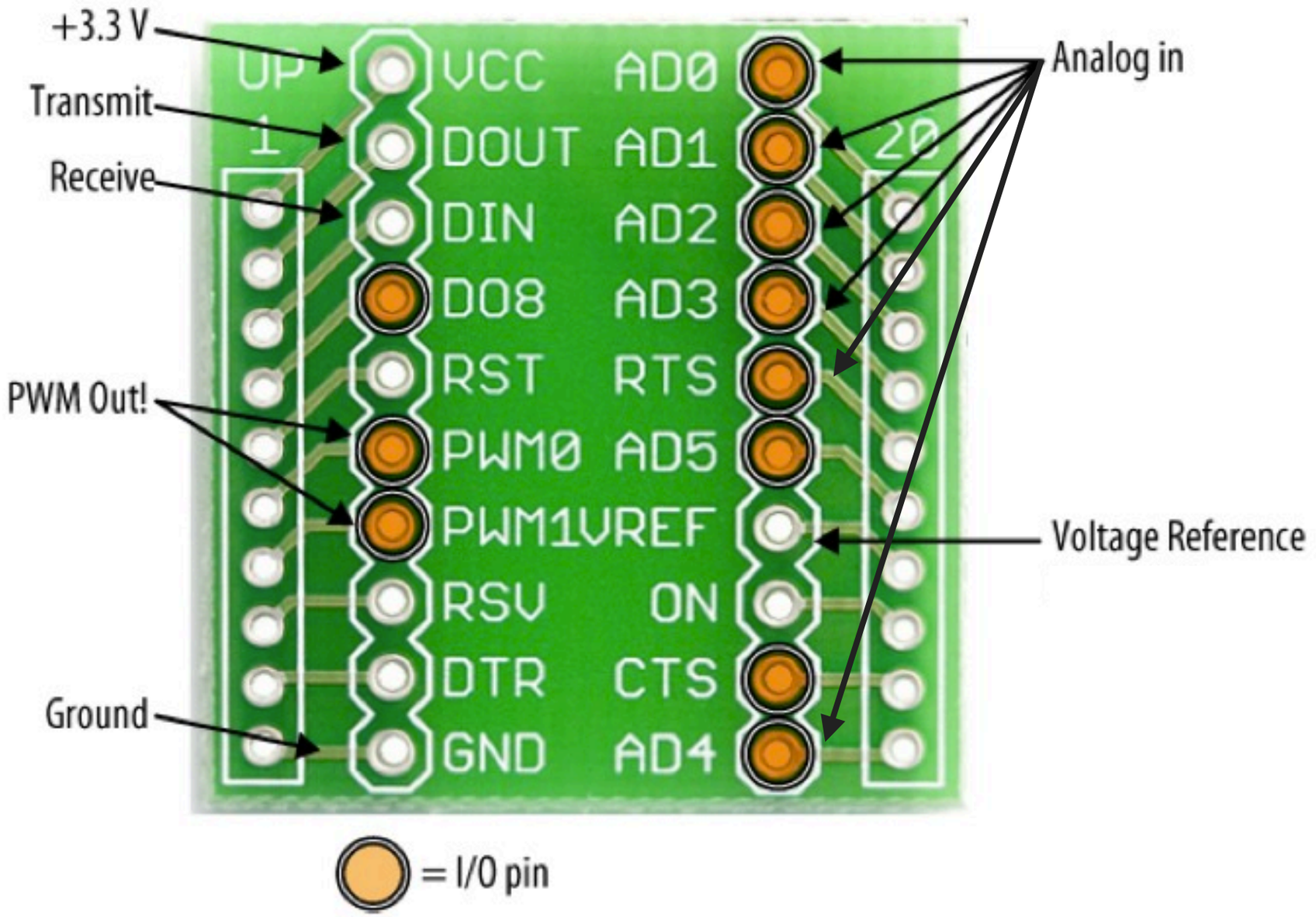
I/O Intro

- For simple input and/or output
- Eight digital input/outputs
- One additional digital output
- Seven analog inputs
- Two analog outputs
- But not all at once! Pins are shared.

I/O Why

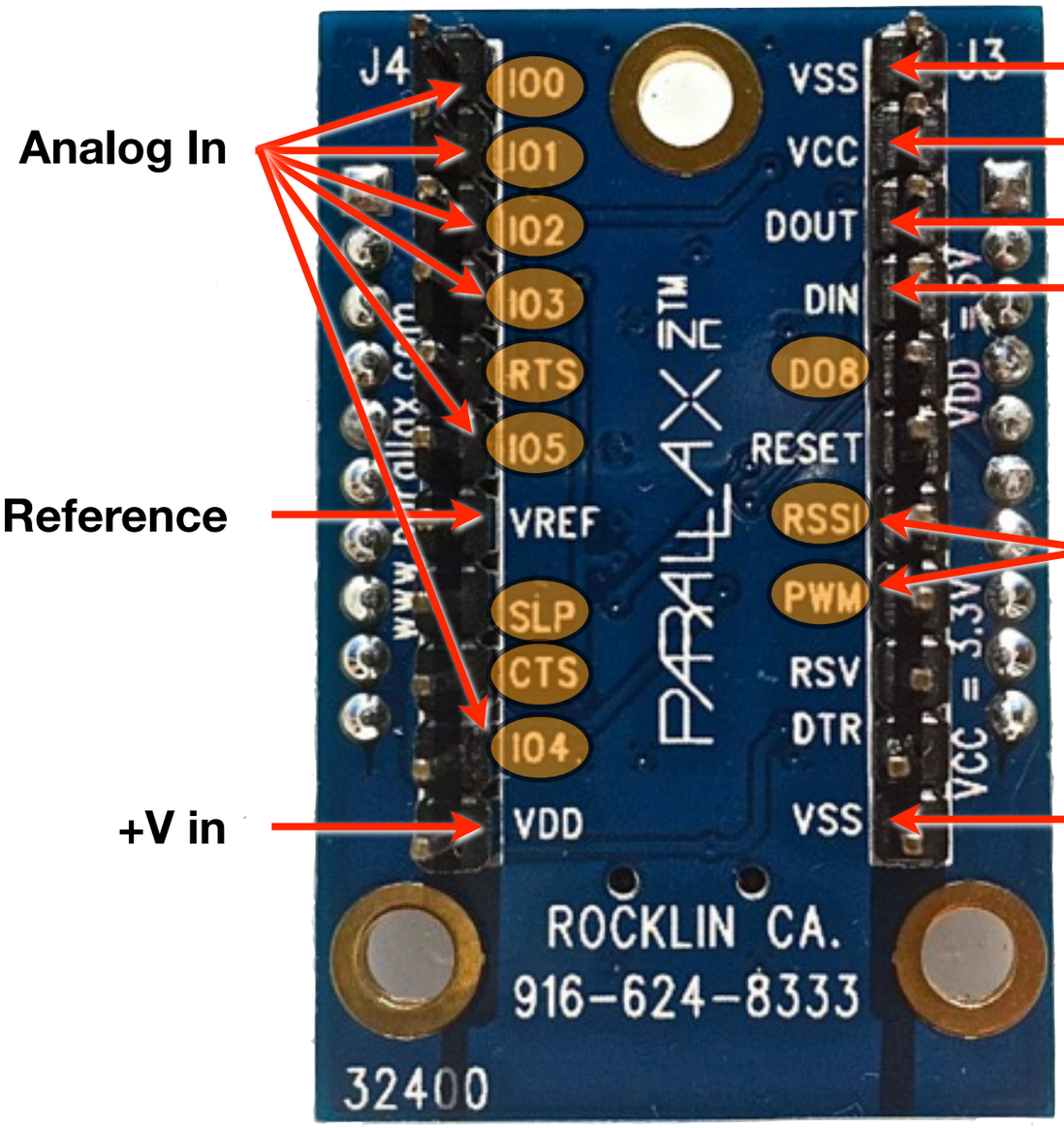
- Why:
  - Save space, save power, save weight and save money
  - Reduce complications for simple projects
- Why not:
  - Limited inputs/outputs
  - No access to logic
  - Might make complicated projects even more complicated

Input/Output Wiring 802.15.4: Basic Breakout



Input/Output Wiring 802.15.4: Parallax XBee USB

= I/O pin



Indicator Lights: Parallax XBee USB

LED FUNCTIONS:

- Yellow – Power
- Green – ON (not sleeping)
- Blue – RSSI (receive data)
- Red – Association Indicator

The USB connector also has two LEDs which indicate TX / RX status:

- Red - transmit to the PC
- Green - receive from the PC

I/O AT Commands

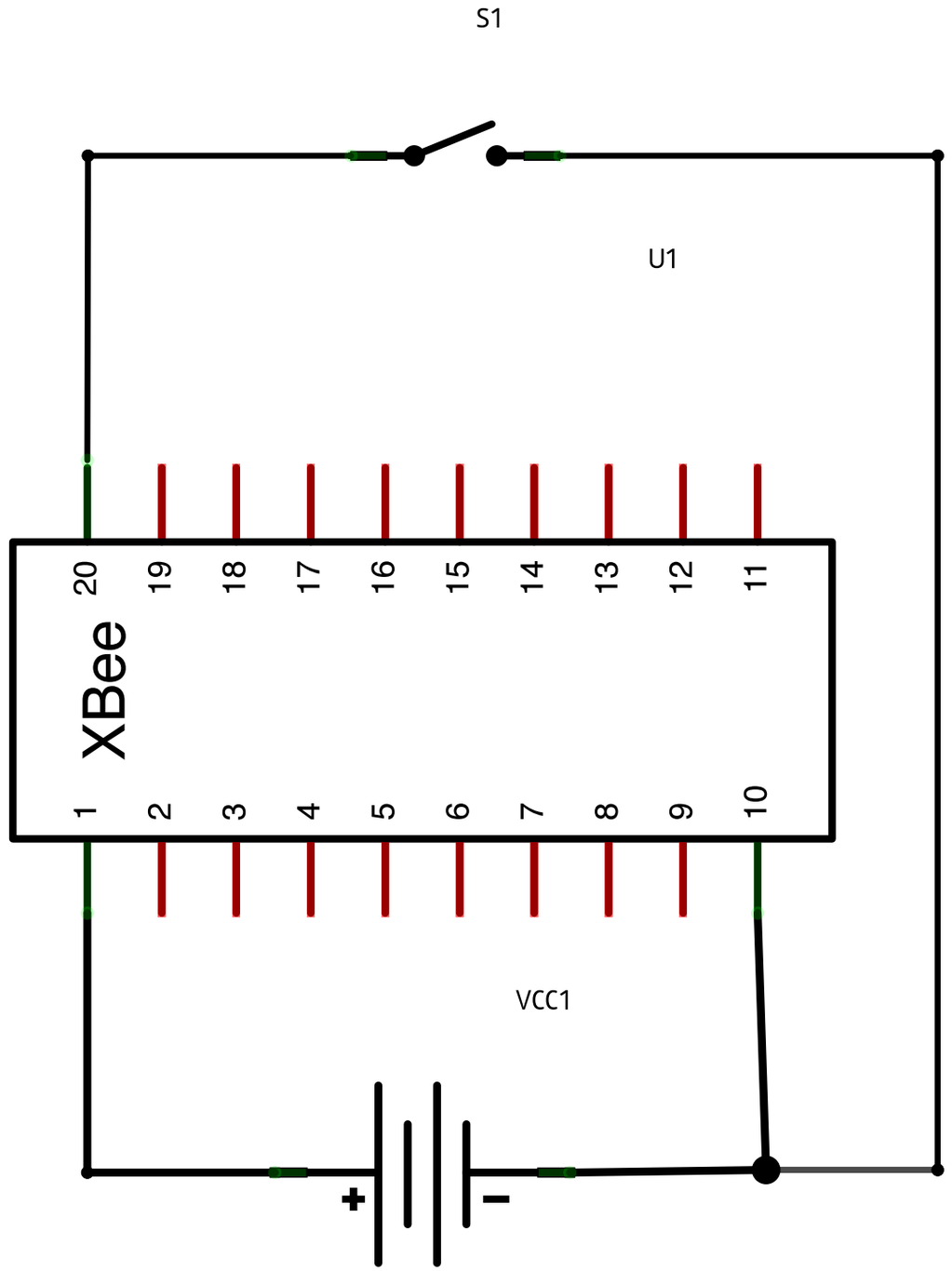
- ATD0...D8 -> configure pins for I/O
- ATIR -> sample rate
- ATIT -> samples before transmit
- ATP0...P1 -> PWM configuration
- ATIA -> I/O input address

Setting I/O Pins

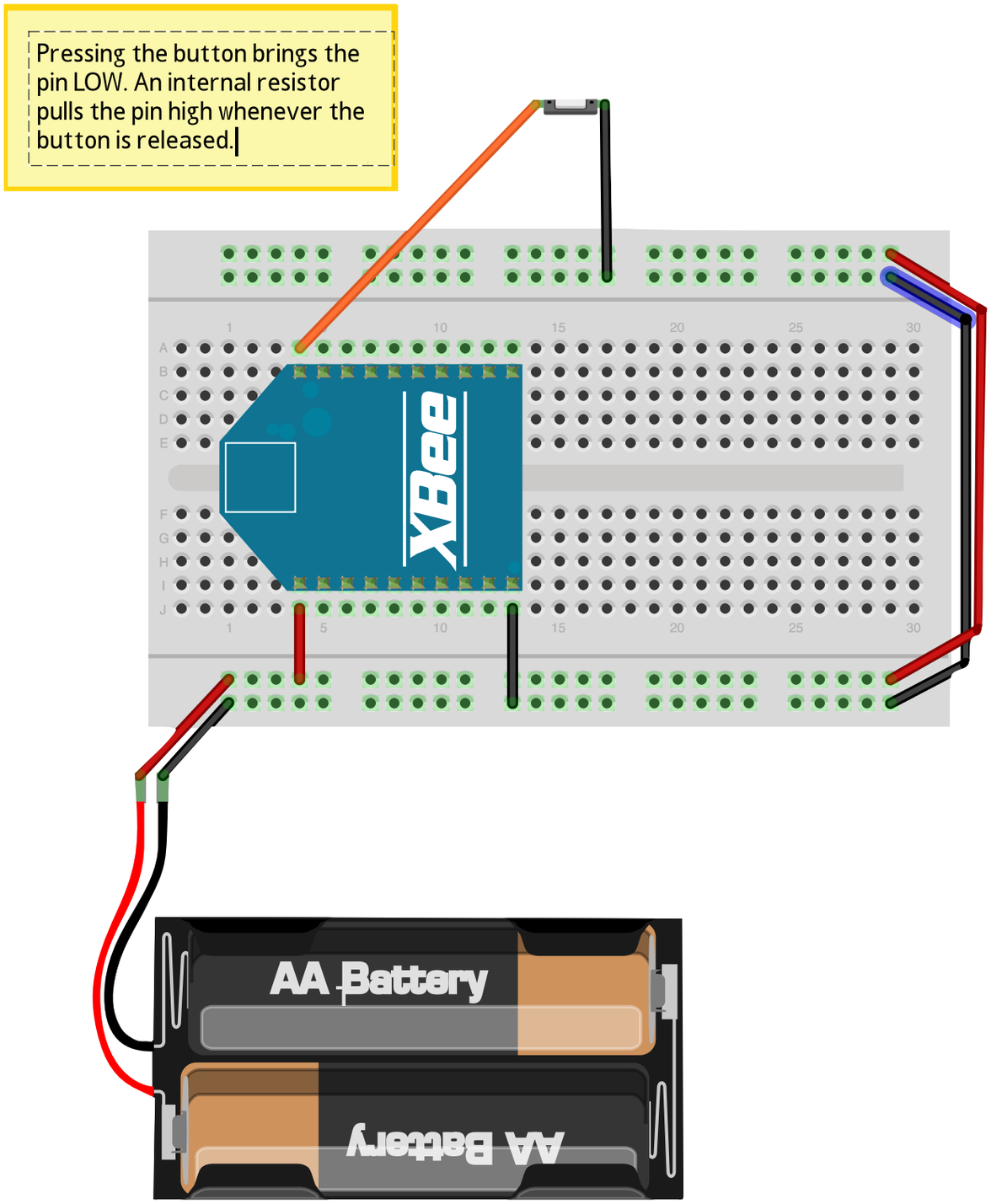
- ATDx 0 Disabled
- ATDx 1 Built-in Function (sometimes)
- ATDx 2 Analog Input (sometimes)
- ATDx 3 Digital Input
- ATDx 4 Digital Output, low to start with
- ATDx 5 Digital Output, high to start with
    - ...so ATD32 would do what?

Basic Doorbell Project

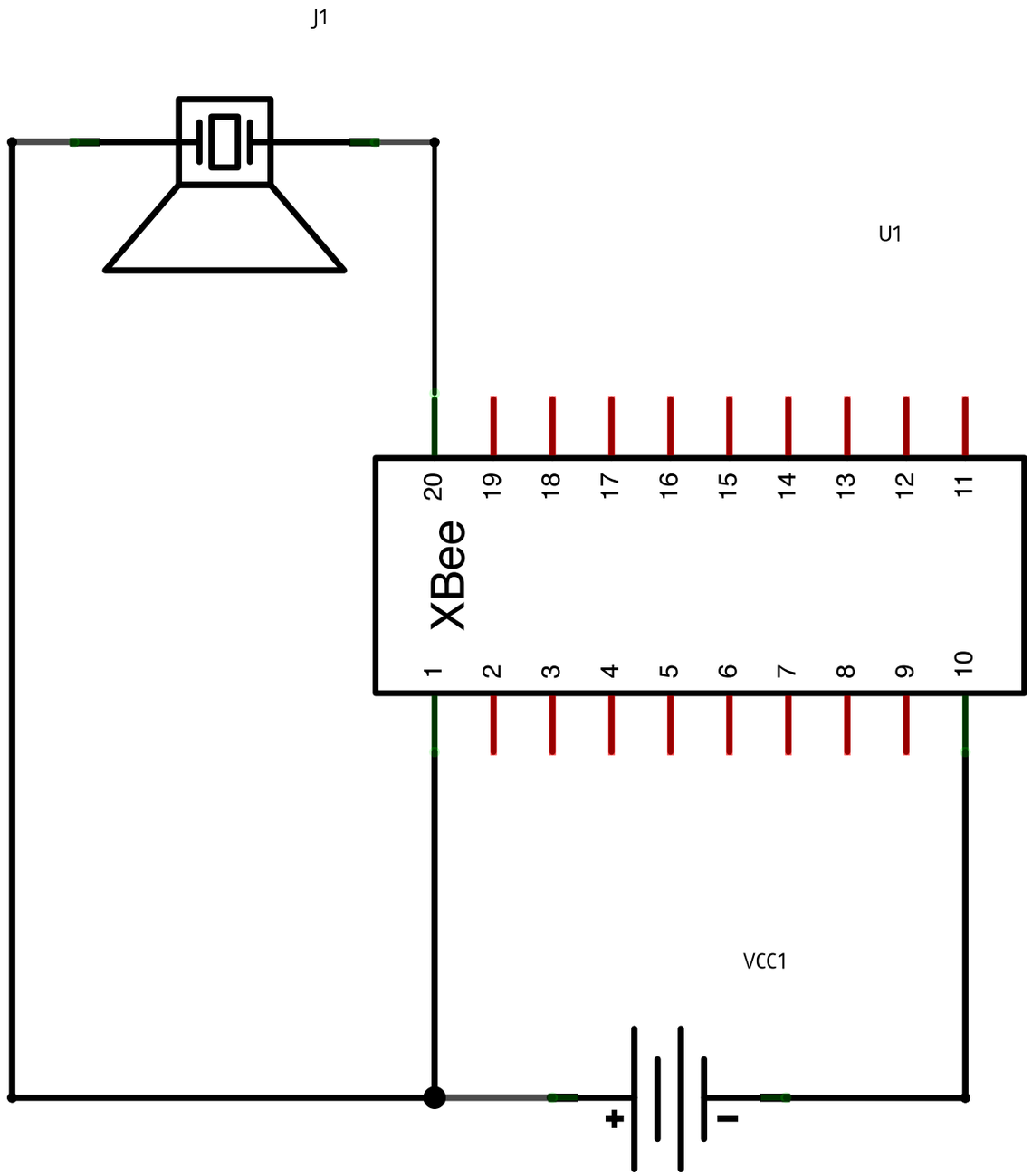
Button Schematic



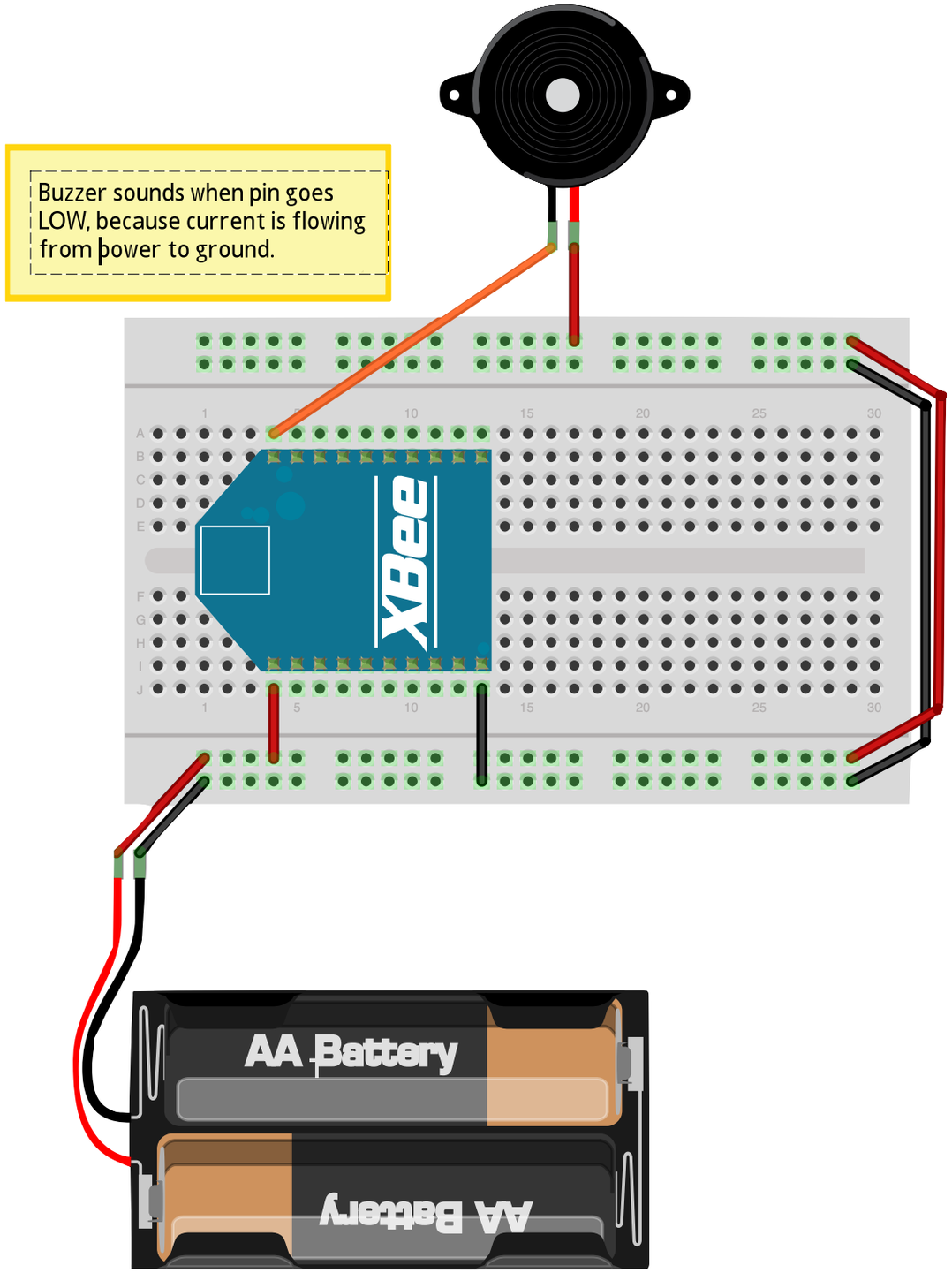
Button Breadboard



Buzzer Schematic



Buzzer Breadboard



Setup Strings

- Button XBee:
    - ATRE,ID*3001*,MY1,DL2,IR64,IT1,D03,IAFFFF,WR

- Buzzer XBee:
    - ATRE,ID*3001*,MY2,DL1,IR64,IT1,D05,IAFFFF,WR
        - *** be sure to change *3001* to your own PAN ID!!

Addressing

- ATRE
  resets to factory settings

- ATID
  sets the PAN ID (choose your own)

- ATMY
  sets the local radio’s address

- ATDL
  sets the destination address

Input/Output Settings

- ATIR
  sets the data sample rate (uses hexadecimal notation)

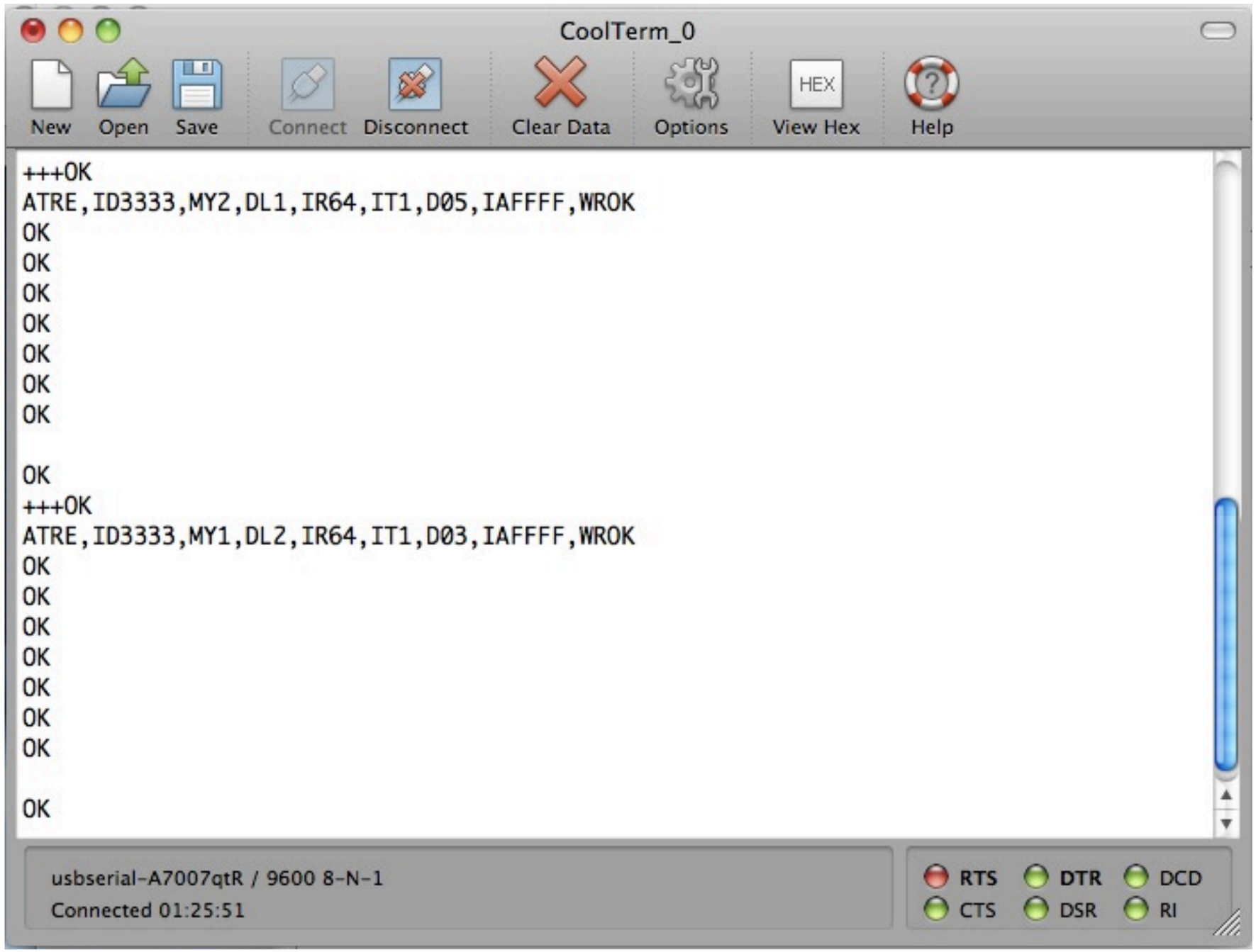
- ATIT
  how many samples transmitted at a time

- ATD0
  mode for digital pin zero (3=digital input, 5=digital output)

- ATIA
  remote address that’s allowed to control local pins

- ATWR
  writes the settings to firmware (like saving to a disk)

CoolTerm


CoolTerm_0
New
Open
Save
Connect
Disconnect
Clear Data
Options
HEX
View Hex
Help
+++OK
ATRE,ID3333,MY2,DL1,IR64,IT1,D05,IAFFFF,WROK
OK
OK
OK
OK
OK
OK
OK
OK
OK
+++OK
ATRE,ID3333,MY1,DL2,IR64,IT1,D03,IAFFFF,WROK
OK
OK
OK
OK
OK
OK
OK
OK
OK
usbserial-A7007qtR / 9600 8-N-1
Connected 01:25:51
RTS
DTR
DCD
CTS
DSR
RI

More

- Got it already?
    - Try going the other way: a light for “I’ll be right there” feedback.
        - remember that input and output pins are paired and mirrored
- Use analog: how loud to ring (use light to simulate if needed)
    - ATD02 sets for analog *inputs*
    - analog *outputs* come from PWM pins ATP0 & ATP1, so paired but *not* mirrored with inputs

ZigBee Addressing

ZigBee Coordinator

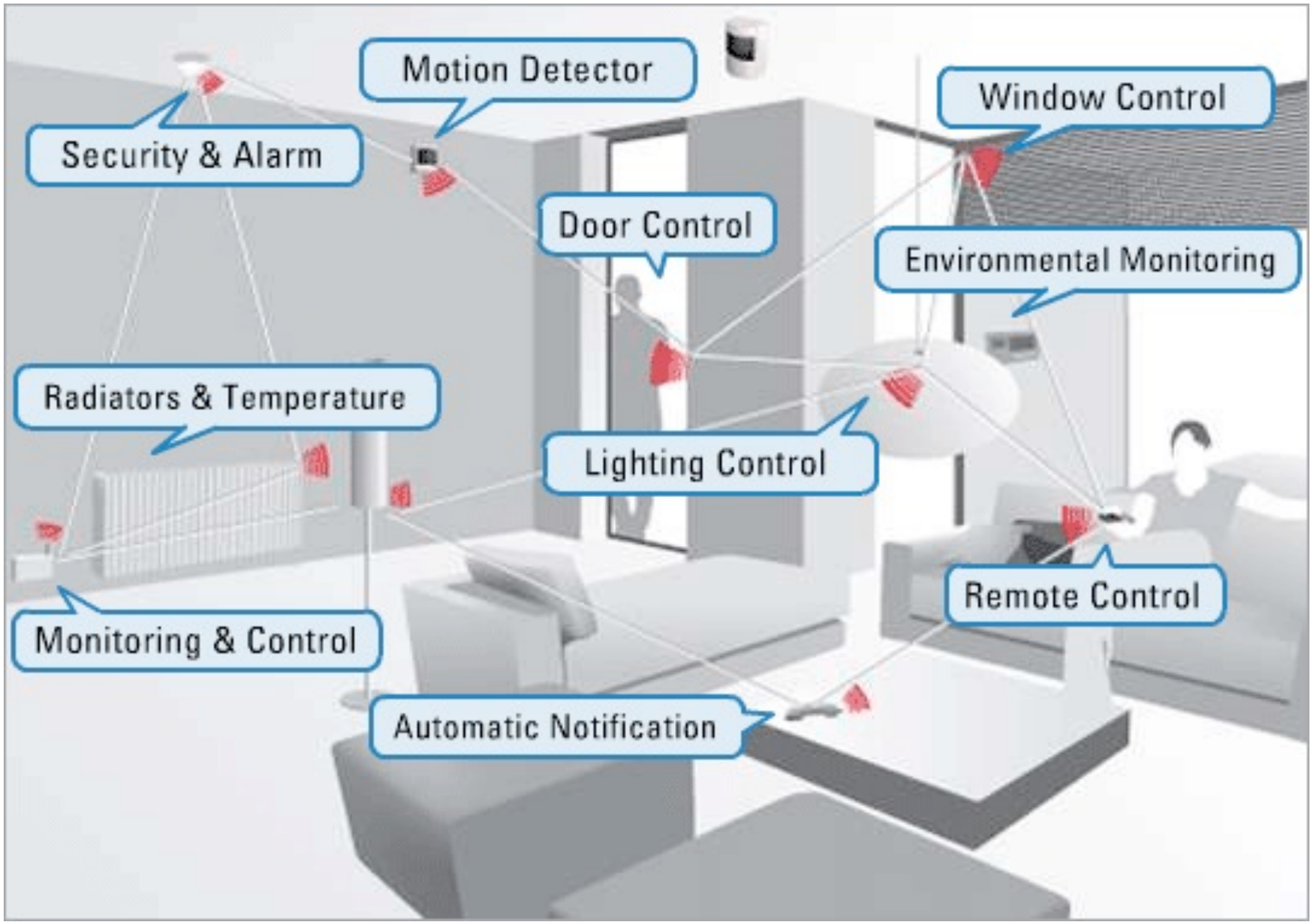
- Every ZigBee network must have a coordinator
- There can only be one coordinator
- Coordinator selects channel and PAN ID
- End devices and routers can then join the PAN
- Typically mains-powered
- Coordinator’s 16-bit address is always 0

ZigBee Router

- Non-coordinator routers are optional to ZigBee networks
- Typically mains-powered
- Many can be on each PAN
- Issues a beacon request on startup to locate channel and PAN
- Routers can communicate with any device on the network
- Stores packets for sleeping end devices
- 16-bit address assigned by coordinator

ZigBee End Device

- *Optional* to ZigBee networks
- Typically battery-powered
- Many can be on each PAN
- Issues a beacon request on startup to locate channel, PAN and parent
- End devices can only communicate directly with their parent
- 16-bit address assigned by coordinator

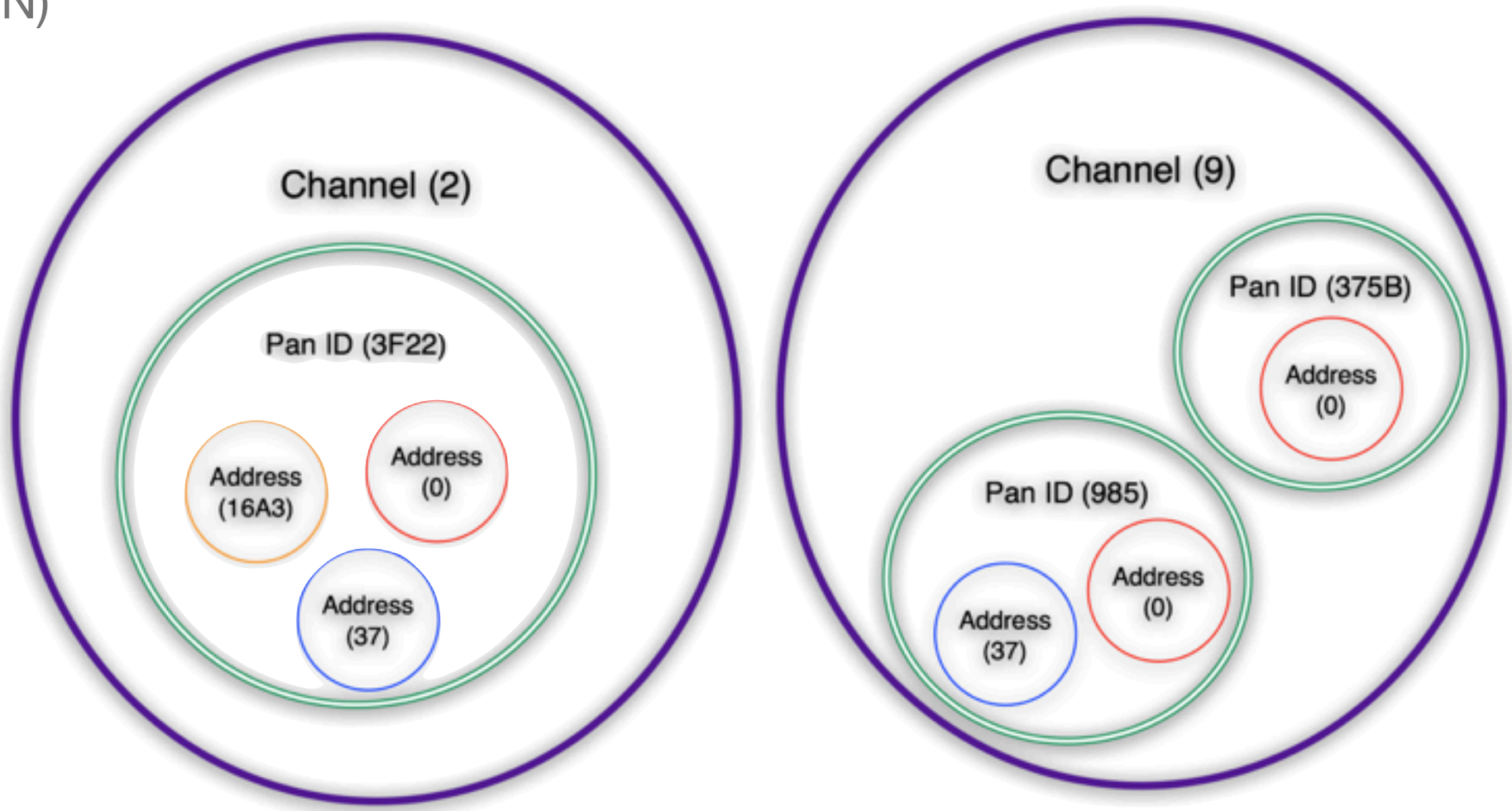

Motion Detector
Security & Alarm
Window Control
Door Control
Environmental Monitoring
Radiators & Temperature
Lighting Control
Remote Control
Monitoring & Control
Automatic Notification

XBee ZB

- Coordinator Firmware
  - for AT commands or API
- Router and End Device Firmware
  - for AT commands or API
- ...so 6 different firmware combinations (you'll always use 2 at the same time)
- and two power levels, regular and Pro
- and 4 antennas! whip, chip, U.FL and RPSMA.

Addressing Basics

- channels
- PAN ID
- 64 bit addresses (SN)
- 16 bit addresses



Firmware Updates

X-CTU

About

PC Settings | Range Test | Terminal | Modem Configuration

Com Port Setup

Select Com Port

MaxStream PKG-U Serial Port...(COM6)

Baud 9600

Flow Control NONE

Data Bits 8

Parity NONE

Stop Bits 1

Test / Query

Host Setup | User Com Ports | Network Interface

API

Enable API

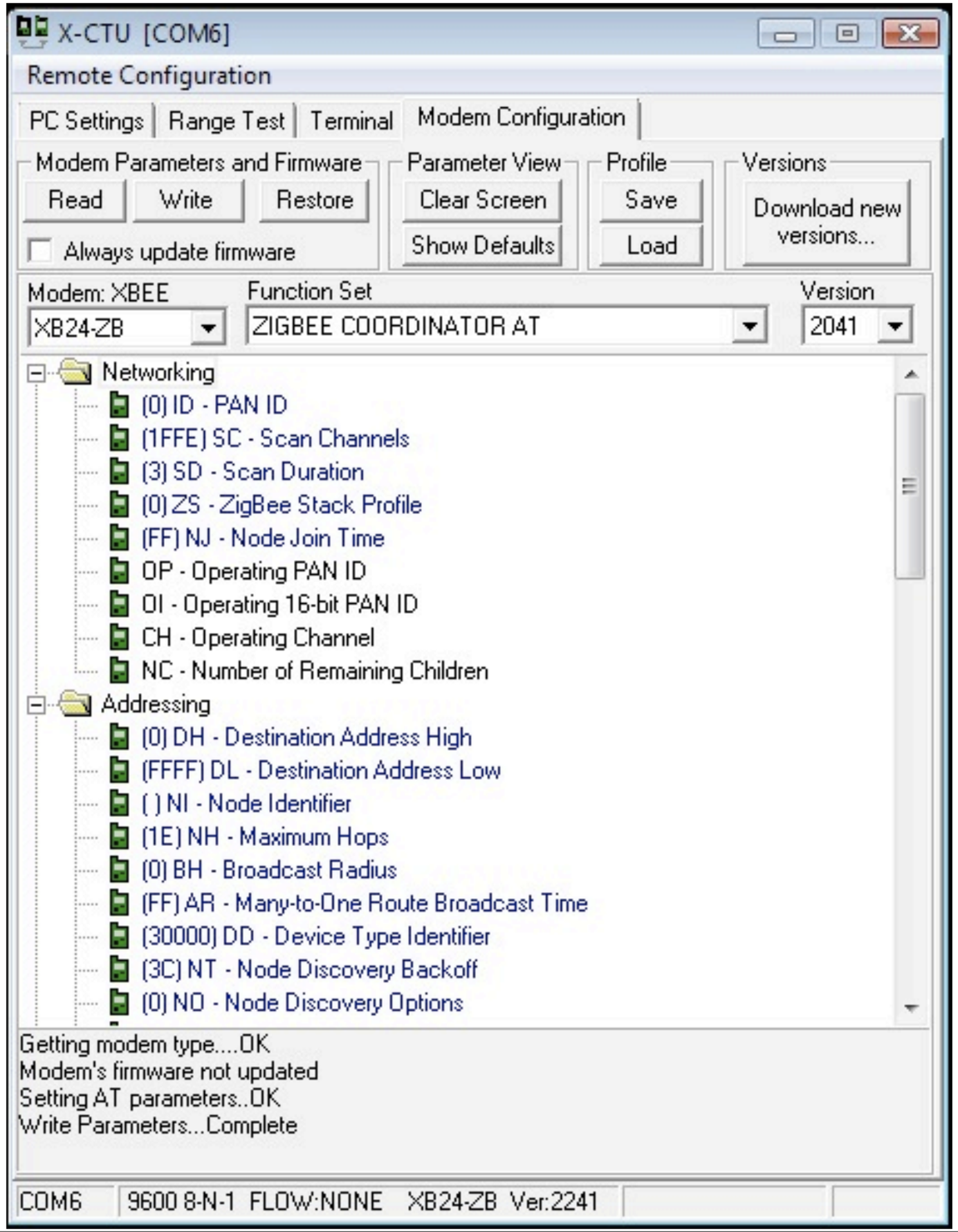
Use escape characters (ATAP = 2)

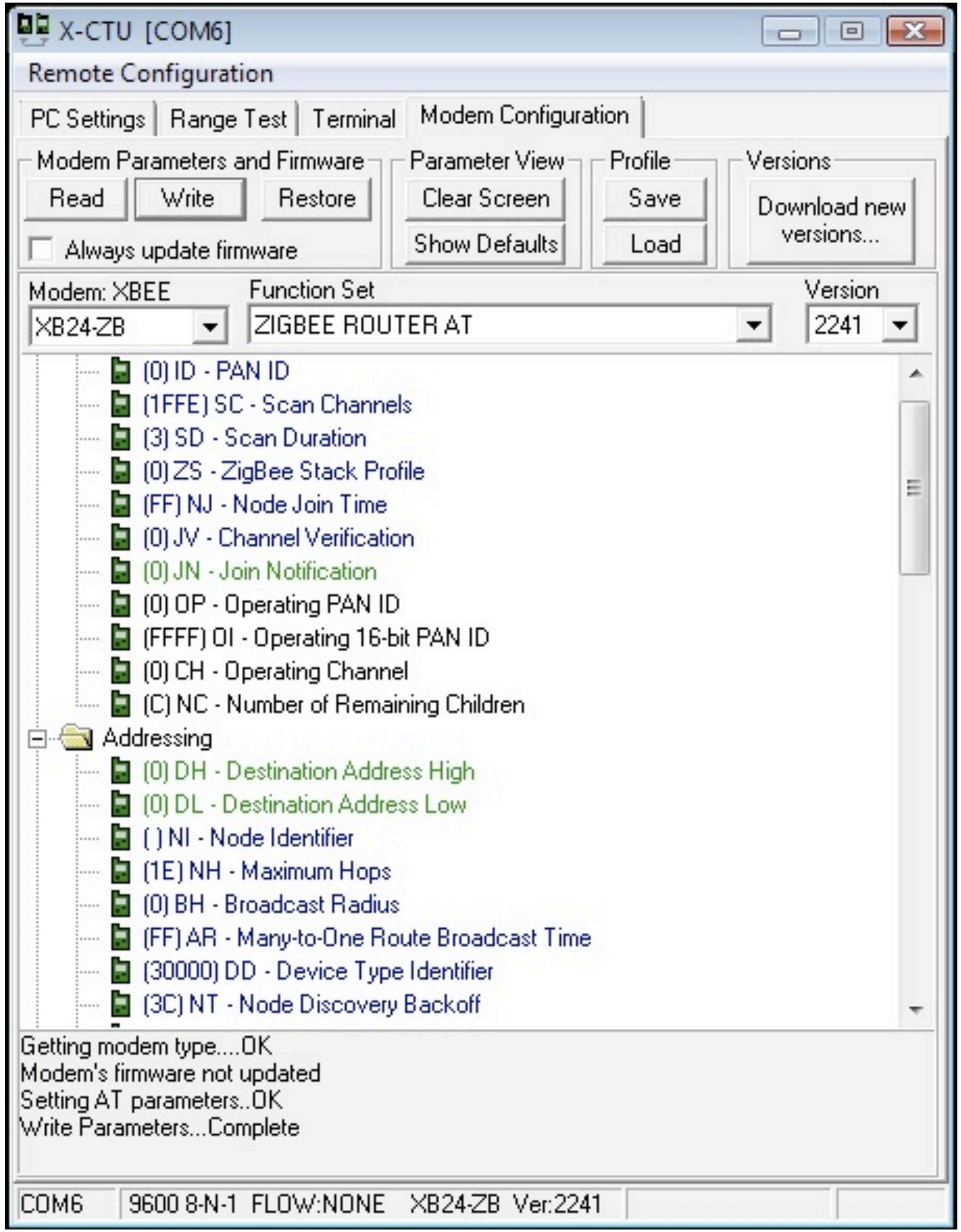
AT command Setup

	ASCII	Hex
Command Character (CC)	+	2B
Guard Time Before (BT)		1000
Guard Time After (AT)		1000

Modem Flash Update

No baud change


X-CTU [COM6]
Remote Configuration
PC Settings
Range Test
Terminal
Modem Configuration
Modem Parameters and Firmware
Read
Write
Restore
Always update firmware
Parameter View
Clear Screen
Show Defaults
Profile
Save
Load
Versions
Download new versions...
Modem: XBEE
XB24-ZB
Function Set
ZIGBEE COORDINATOR AT
Version
2041
Networking
(0) ID - PAN ID
(1FFE) SC - Scan Channels
(3) SD - Scan Duration
(0) ZS - ZigBee Stack Profile
(FF) NJ - Node Join Time
OP - Operating PAN ID
OI - Operating 16-bit PAN ID
CH - Operating Channel
NC - Number of Remaining Children
Addressing
(0) DH - Destination Address High
(FFFF) DL - Destination Address Low
( ) NI - Node Identifier
(1E) NH - Maximum Hops
(0) BH - Broadcast Radius
(FF) AR - Many-to-One Route Broadcast Time
(30000) DD - Device Type Identifier
(3C) NT - Node Discovery Backoff
(0) NO - Node Discovery Options
Getting modem type....OK
Modem's firmware not updated
Setting AT parameters..OK
Write Parameters...Complete
COM6
9600 8-N-1 FLOW:NONE XB24-ZB Ver:2241


X-CTU [COM6]
Remote Configuration
PC Settings
Range Test
Terminal
Modem Configuration
Modem Parameters and Firmware
Read
Write
Restore
Always update firmware
Parameter View
Clear Screen
Show Defaults
Profile
Save
Load
Versions
Download new versions...
Modem: XBEE
XB24-ZB
Function Set
ZIGBEE ROUTER AT
Version
2241
(0) ID - PAN ID
(1FFE) SC - Scan Channels
(3) SD - Scan Duration
(0) ZS - ZigBee Stack Profile
(FF) NJ - Node Join Time
(0) JV - Channel Verification
(0) JN - Join Notification
(0) OP - Operating PAN ID
(FFFF) OI - Operating 16-bit PAN ID
(0) CH - Operating Channel
(C) NC - Number of Remaining Children
Addressing
(0) DH - Destination Address High
(0) DL - Destination Address Low
( ) NI - Node Identifier
(1E) NH - Maximum Hops
(0) BH - Broadcast Radius
(FF) AR - Many-to-One Route Broadcast Time
(30000) DD - Device Type Identifier
(3C) NT - Node Discovery Backoff
Getting modem type....OK
Modem's firmware not updated
Setting AT parameters..OK
Write Parameters...Complete
COM6
9600 8-N-1 FLOW:NONE XB24-ZB Ver:2241

Basic Configuration

Download and Install Software & Drivers

- Download & install the FTDI USB drivers:
  http://www.ftdichip.com/Drivers/VCP.htm

- Download the CoolTerm:
  http://freeware.the-meiers.org/

Other Serial Terminal Options:
settings: 9600 baud, 8 bits, no parity, one stop bit, no flow control

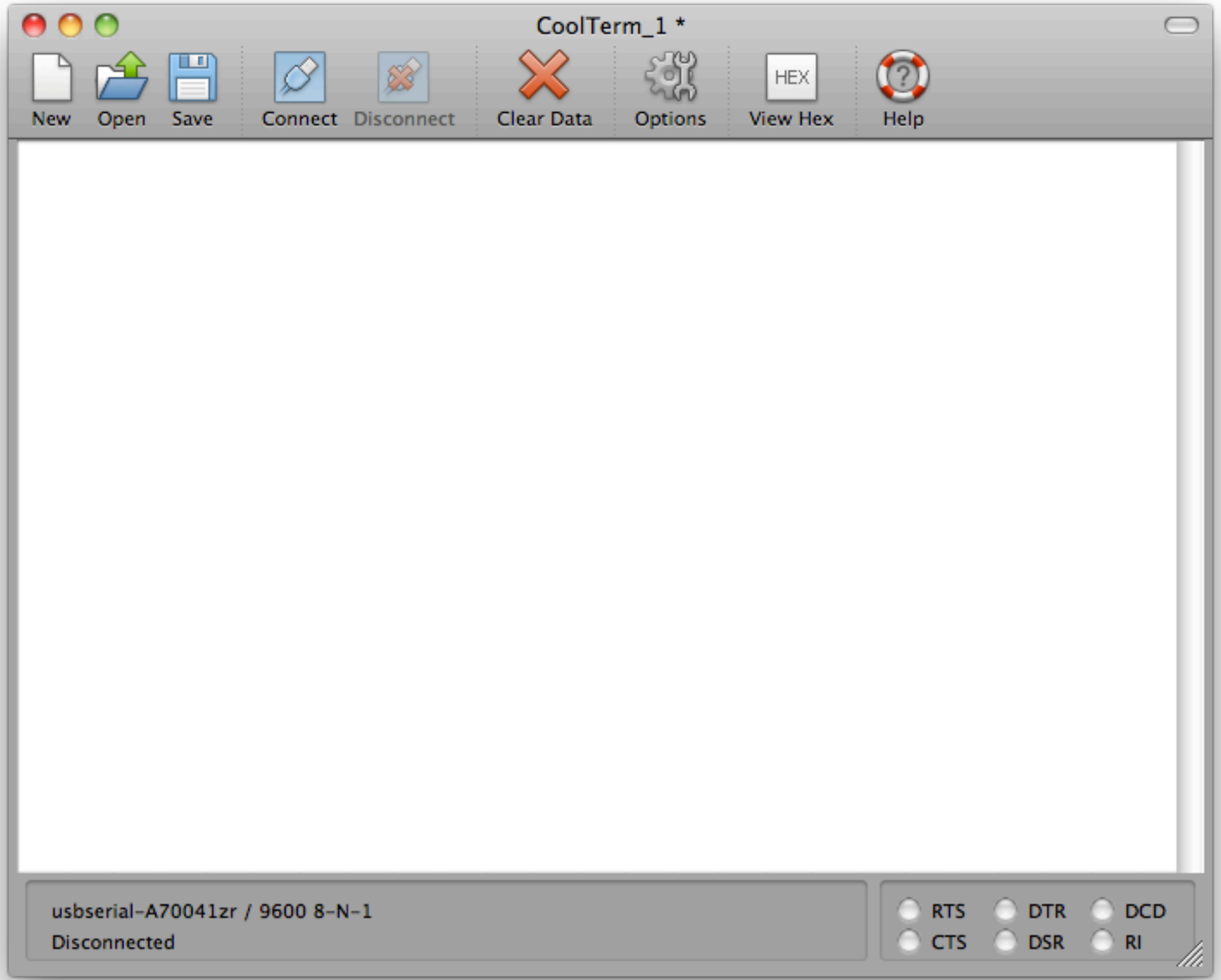
- X-CTU: http://www.digi.com/support/productdetl.jsp?pid=3352&osvid=57&tp=4&s=316

- Z-Term: http://homepage.mac.com/dalverson/zterm/

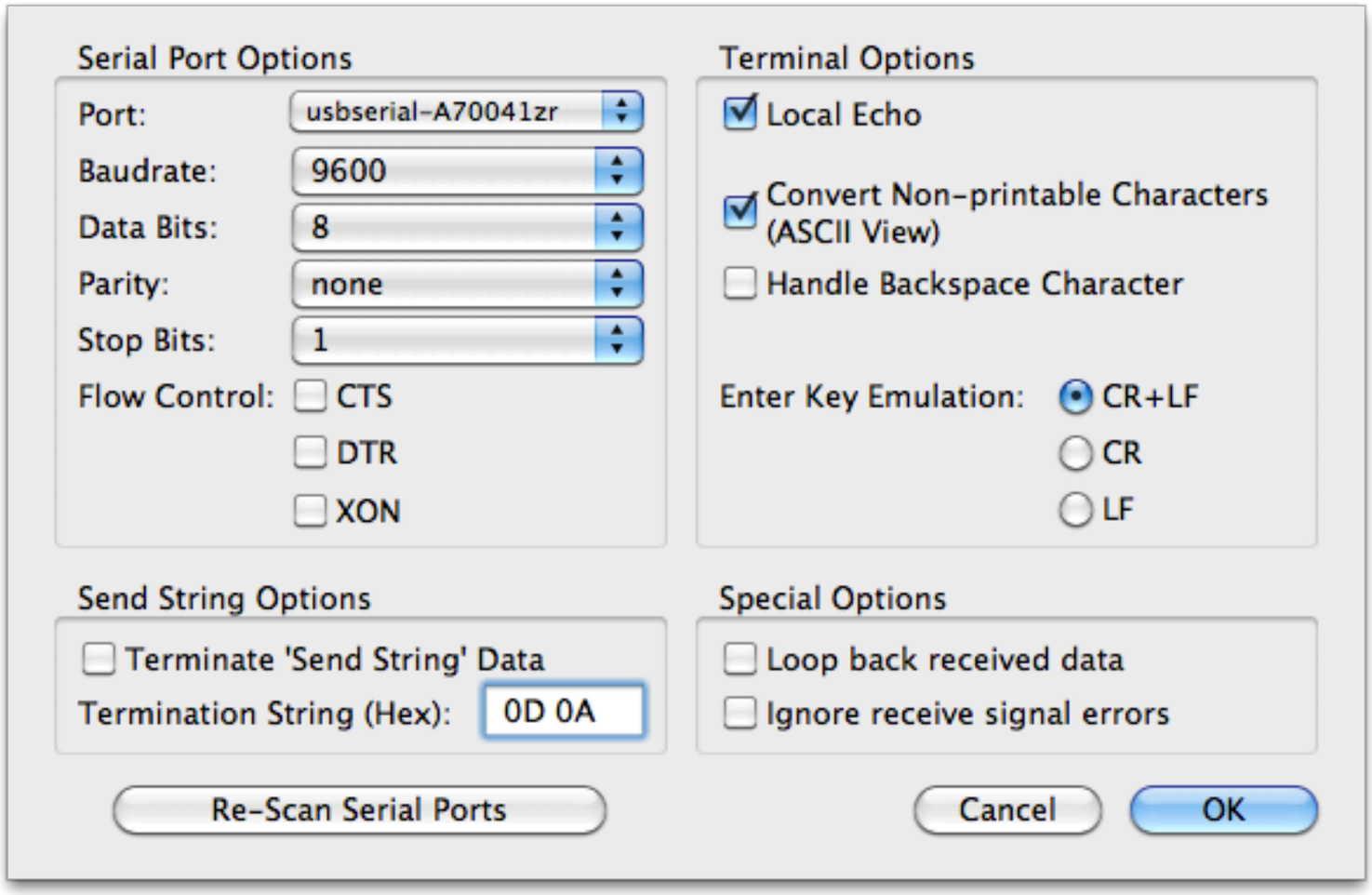
- HyperTerm: Windows Start Menu, Accessories, Communication

- Screen: Terminal program on the Mac (or Linux)

Open CoolTerm

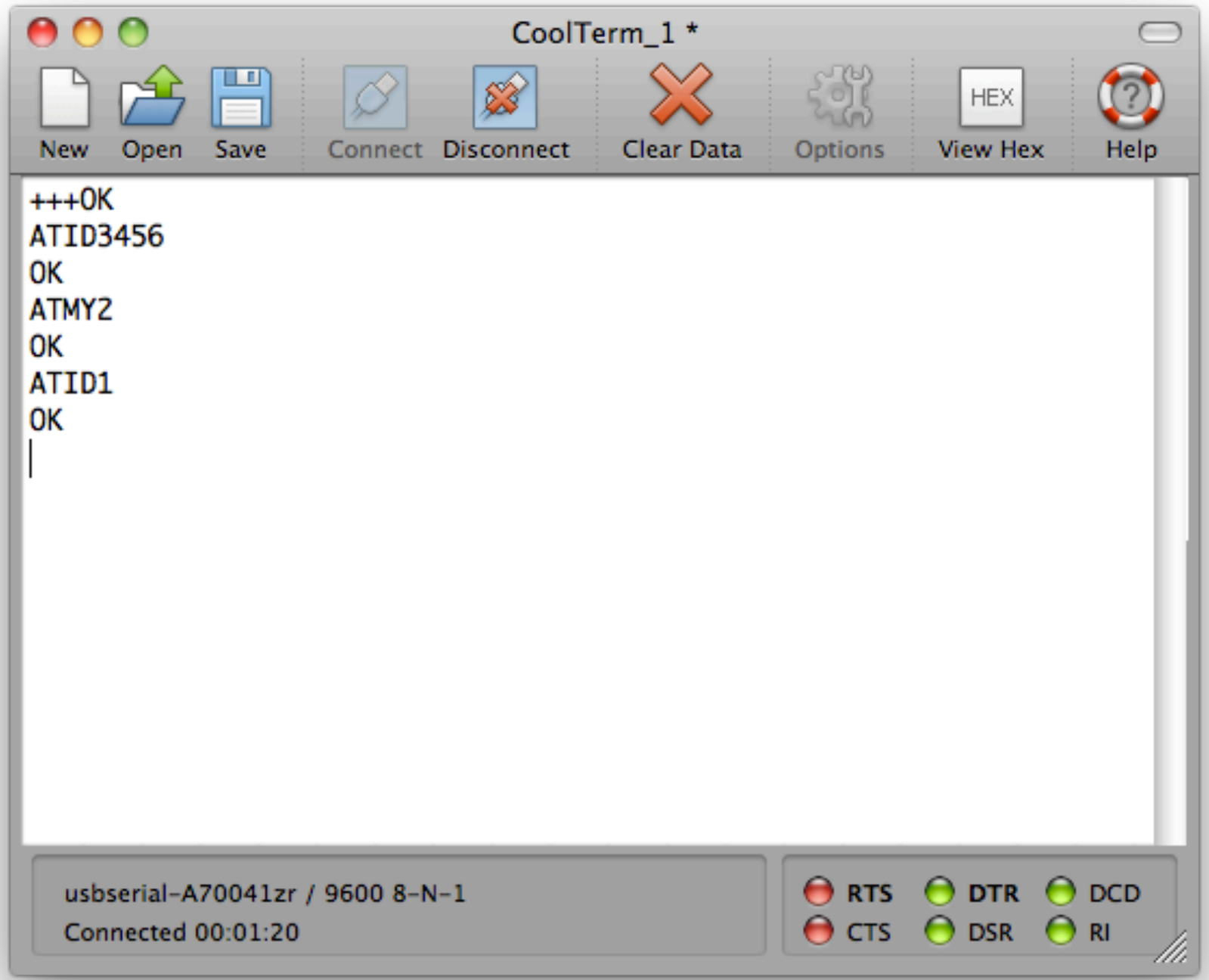


Set Connection Options



Configure your radio with AT commands

- Configure your radio



AT Commands

Some AT Commands

- AT -> OK
- ATDH, ATDL -> destination address hi/lo
- ATID -> personal area network ID
- ATCN -> end command mode
- ATWR -> write current configuration to firmware
- *ATMY* -> ~~my address~~ NOT SETTABLE FOR ZIGBEE
- ATRE -> reset to factory defaults

Pair Exercise

Create a Basic ZigBee Pair

- One coordinator and one router
- Use the 64-bit addresses for destinations
- ATNR will reset your network layer, useful if you join the wrong ID
- Remember, the radios work reliably, troubleshooting is mostly about figuring out what they’re doing.

ZigBee and Arduino

Why Arduino

- local logic
- pinouts
- fast prototyping
- one side of I/O

Arduino Serial Library

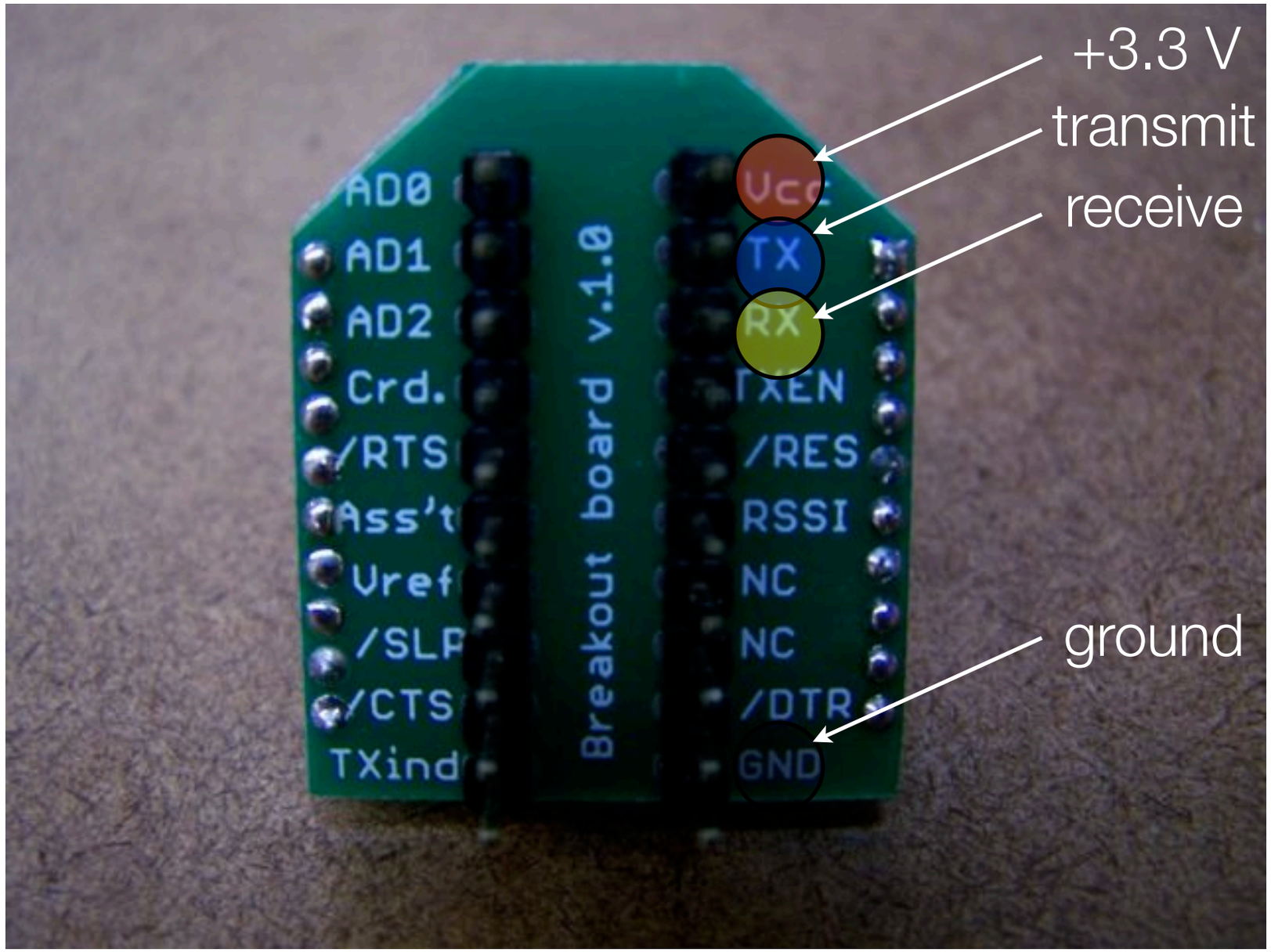
- Serial.begin(speed)
- Serial.available()
- Serial.read()
- Serial.flush()
- Serial.print(data)

Software Serial

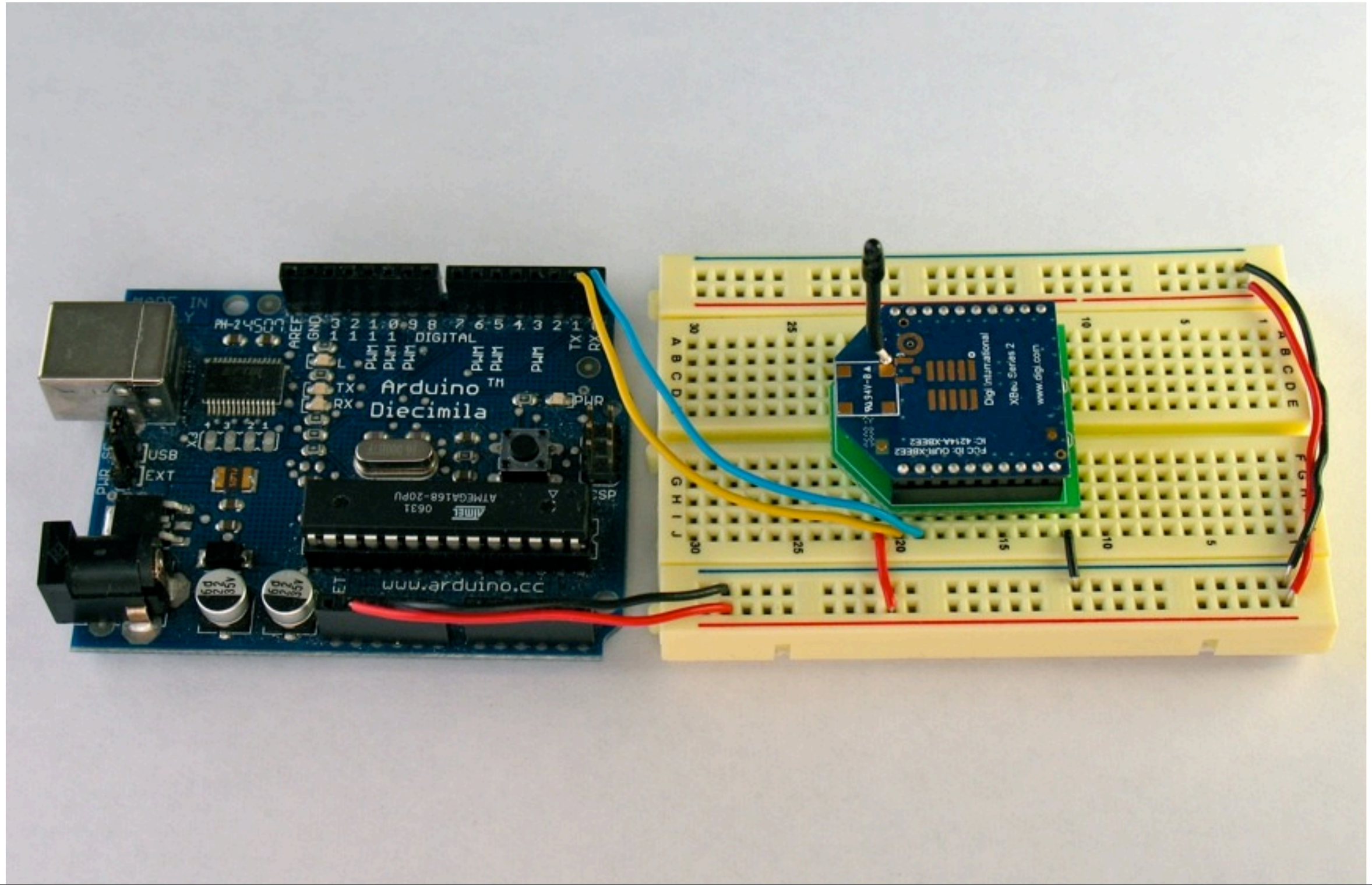
- 115K baud max, all pins are okay to use, all functions available
- buffering!
- good choice for input when you want debug on the HW port for ease-of-use
- in versions *prior* to Arduino 1.0 use:
  http://arduiniana.org/libraries/NewSoftSerial/

Breadboard Hookups

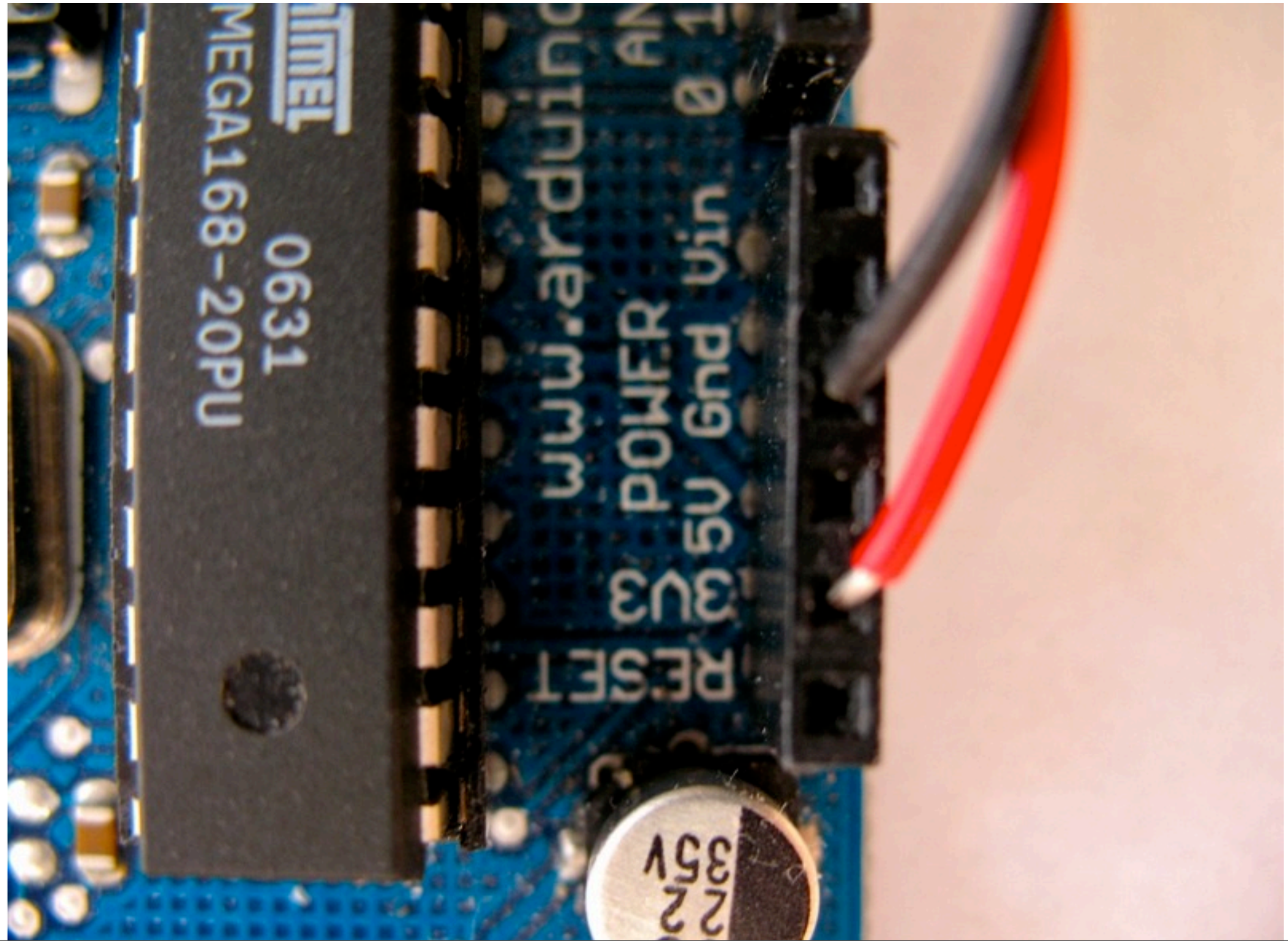
Wiring


+3.3 V
transmit
receive
ground
AD0
AD1
AD2
Crd.
/RTS
Ass't
Vref
/SLP
/CTS
TXind
v.1.0
Breakout board
Vcc
TX
RX
TXEN
/RES
RSSI
NC
NC
/DTR
GND

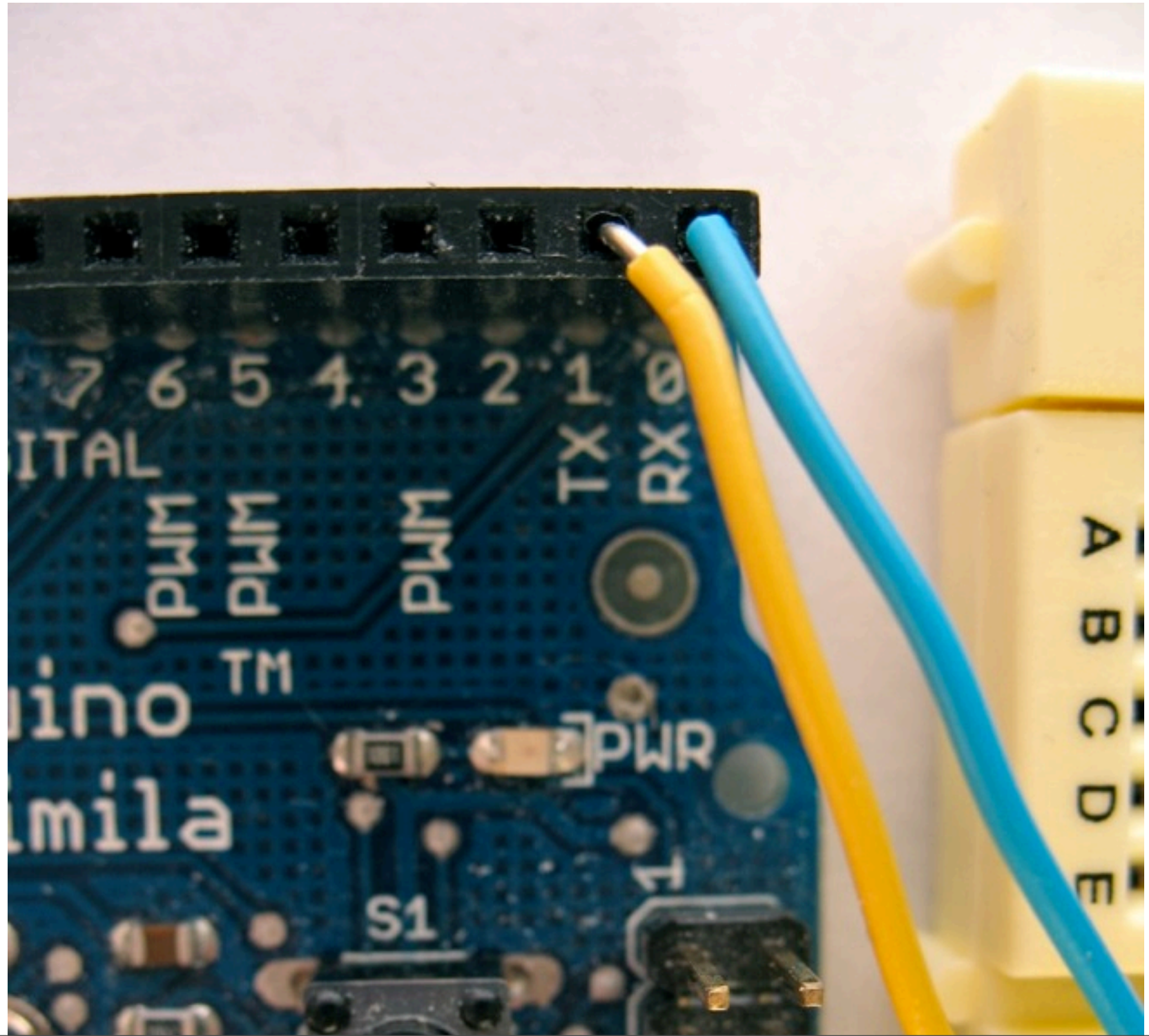
XBee Arduino Breadboard Layout



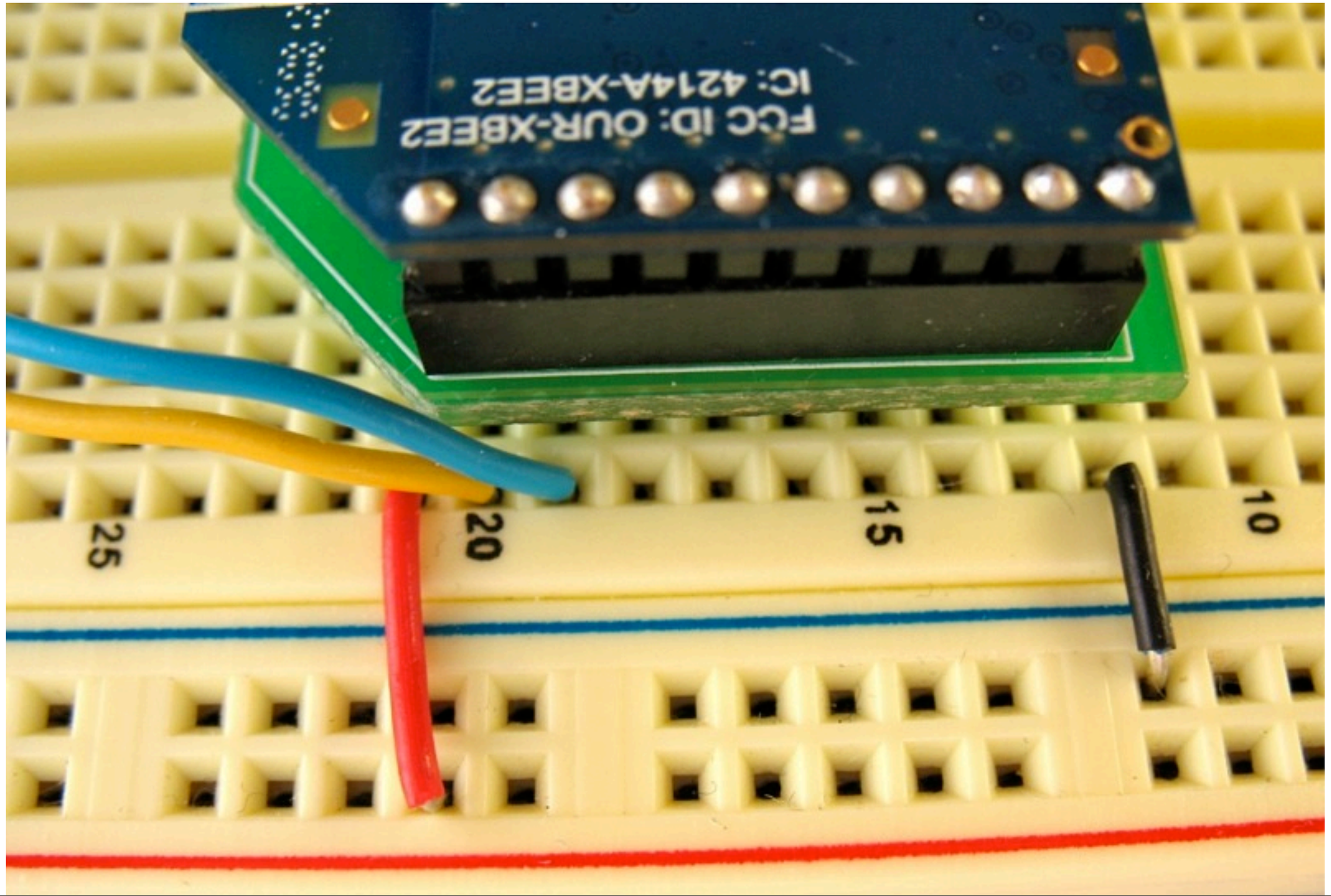
Power, Ground



TX, RX



XBee Connections (pin 1, 2, 3 and 10)



Remember!

- Use only +3.3 Volts. More than +7 Volts will kill your radio
- If you use a voltage regulator, always use decoupling capacitors. The radios often don't work without them.
- XBee TX goes to Arduino RX and vice versa.
- Unplug the TX & RX before uploading Arduino code (or use switches)
- You can't send infinitely fast. Try putting a 10 ms delay into your loop.

I/O Mode

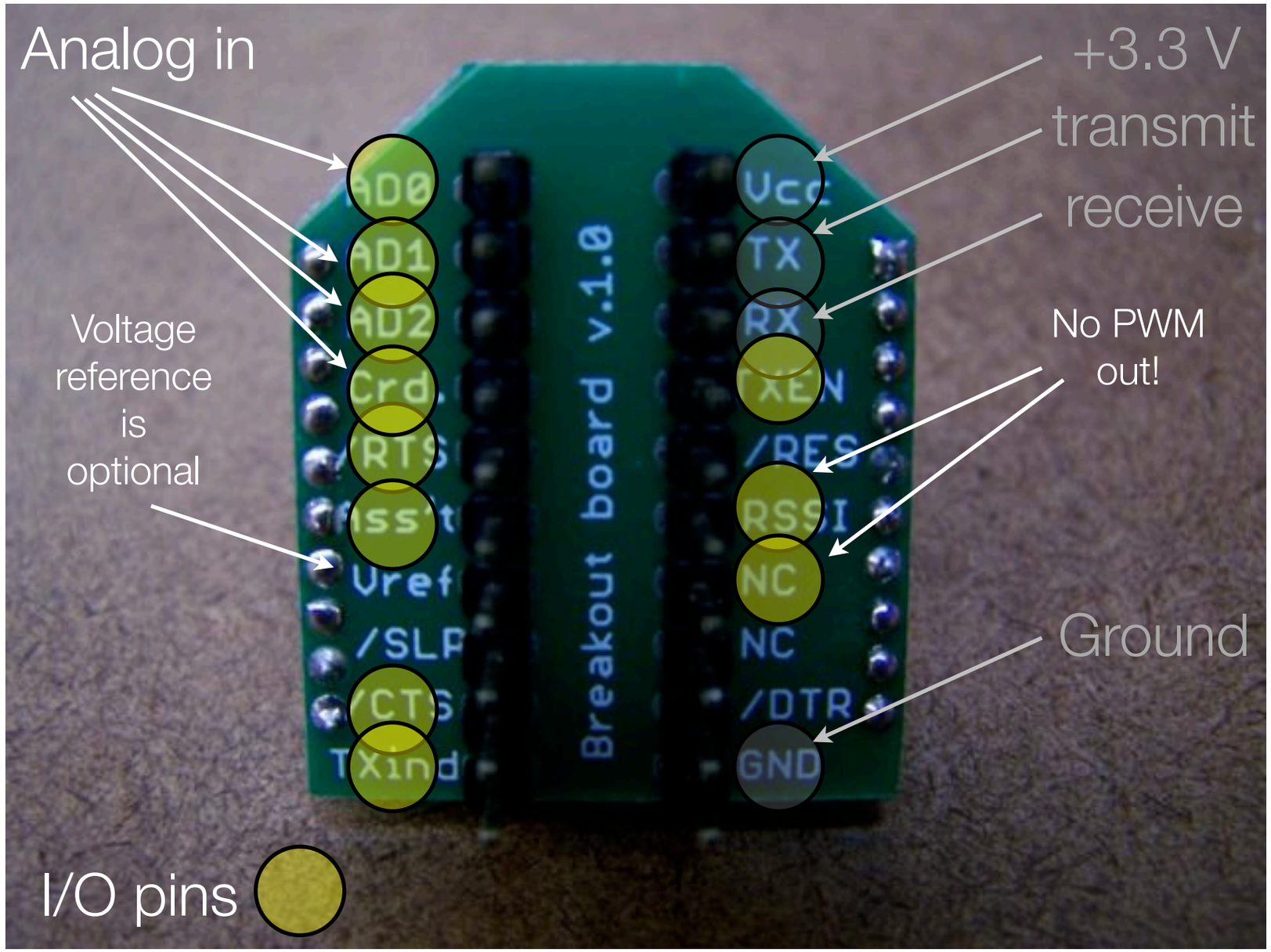
I/O Intro: ZigBee

- For simple input and/or output
- Ten digital input/outputs
- Four analog inputs
- No analog outputs on ZigBee
- But not all at once! Pins are shared.

I/O Why

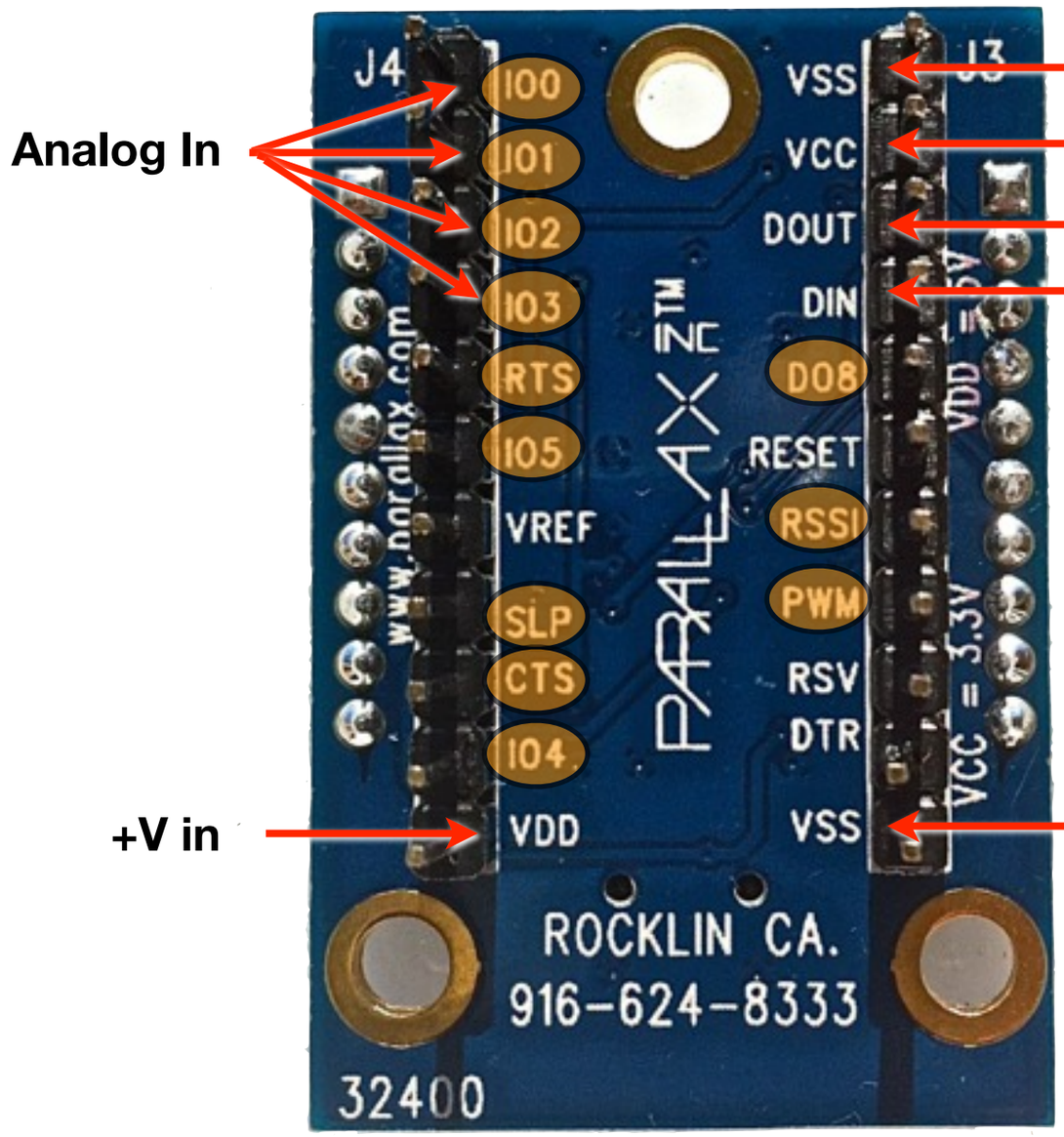
- Why:
  - Save space, save power, save weight and save money
  - Reduce complications
- Why not:
  - Limited inputs/outputs
  - No access to logic
  - No analog output on ZigBee radios

Input/Output Wiring: ZigBee



Input/Output Wiring ZigBee: Parallax XBee USB

= I/O pin



Analog In

Ground

+3.3V

Transmit

Receive

+V in

Ground

I/O AT Commands: ZigBee

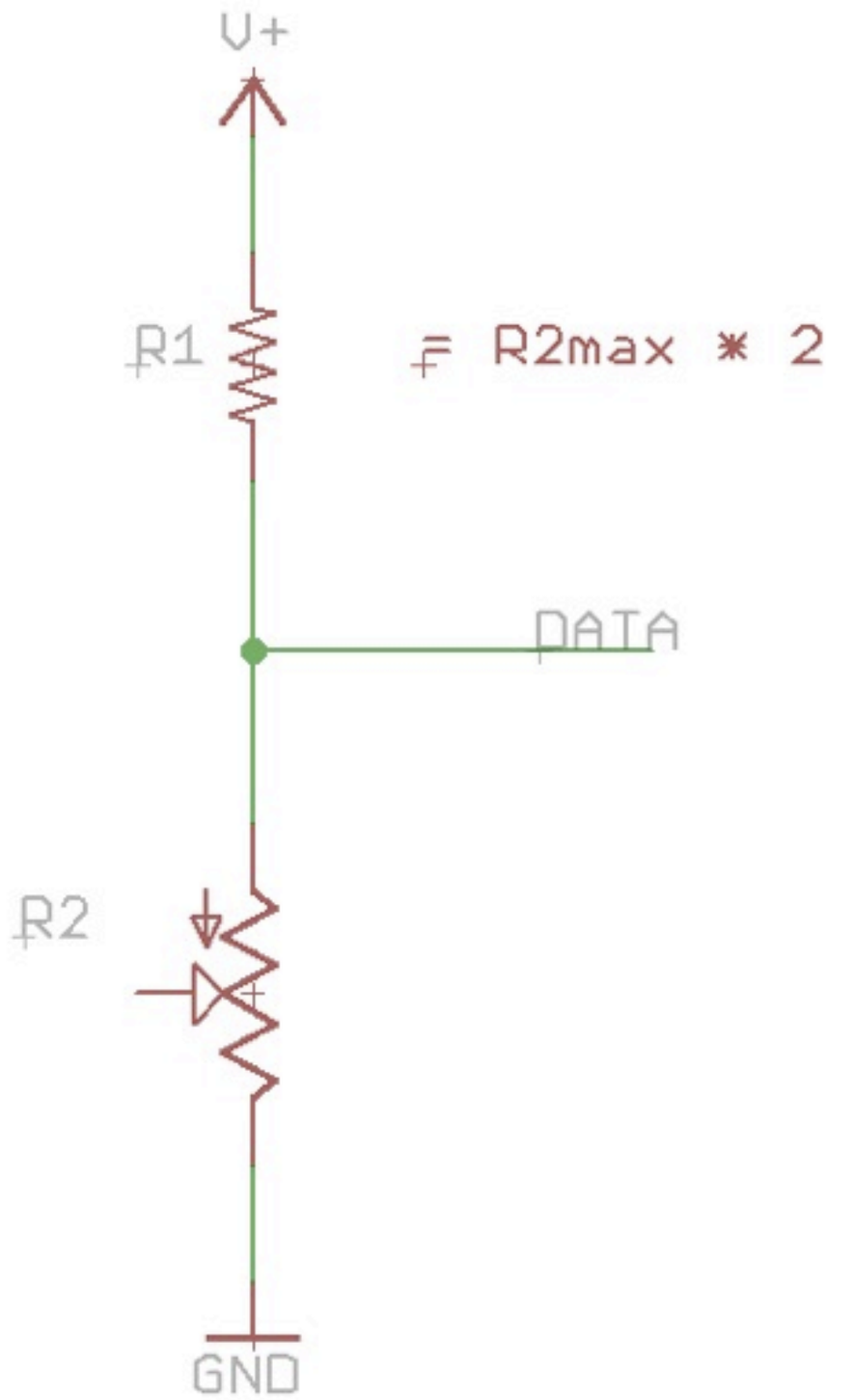
- ATD0...D7 -> configure pins for I/O (D8 and D9 not supported yet)
- ATP0...P1 -> configure pins 10 - 11 for I/O (P3 not supported yet)
- ATIR -> sample rate
- samples before transmit is always 1
- destination address receives sample info
- ALL PINS READ BETWEEN 0 AND 1.2 VOLTS ONLY

Settting I/O Pins

- ATDx 0 Disabled
- ATDx 1 Built-in Function (sometimes)
- ATDx 2 Analog Input (sometimes)
- ATDx 3 Digital Input
- ATDx 4 Digital Output, low to start with
- ATDx 5 Digital Output, high to start with
    - ...so ATD43 would set what?

XBee ZigBees inputs are 1.2V range

Voltage Divider to map 3.3V range to 1.2V range



API Mode Overview

API Mode

- Application Programming Interface
  - “An application programming interface (API) is a source code interface that an operating system or library provides to support requests for services to be made of it by computer programs.”

    http://www.computerworld.com/action/article.do?command=viewArticleBasic&articleId=43487

- XBees in API mode are ready to talk to computers and microcontrollers
  - structured
  - predictable
  - reliable

API Structure

- Used in serial communications with the XBee radio
- Frames of data
  - envelope structure contains data with metadata inside a constrained format
- Radio must be in API Mode
  - AT command ATAP 1 on Series 1 radios
  - API firmware on Series 2 radios

Why API

- Rather than:

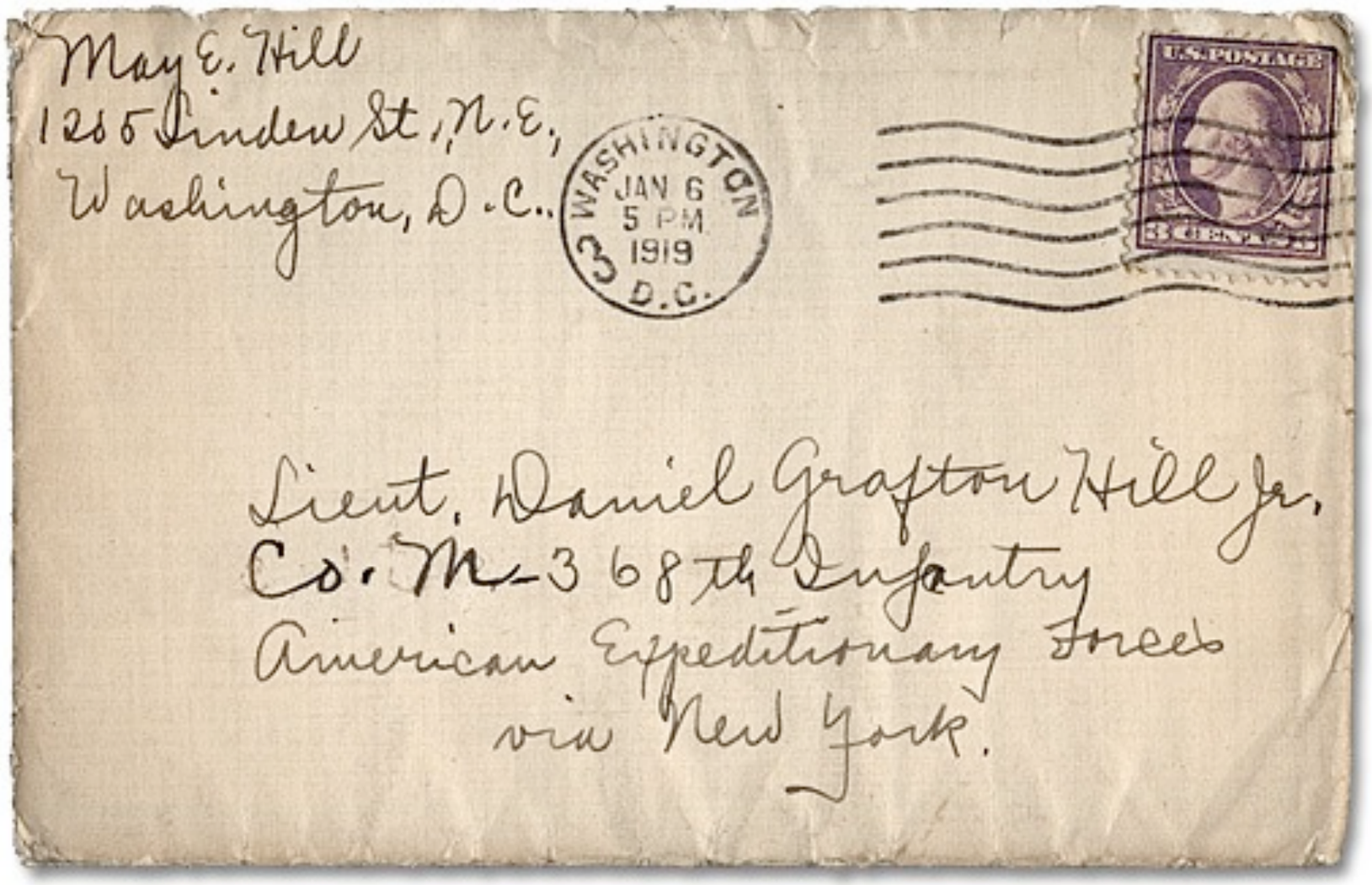
```
delay(1100);
// put the XBee in command mode
Serial.print("+++");
delay(1100);
if (checkFor("OK", 1000)) {
  Serial.println("ATID7777,CN");
   if (checkFor("OK", 1000)) {
      // if an OK was received then continue
      debugPrintln("SetupOK");
      success = true;
  }
}
```

- With a library you just write:

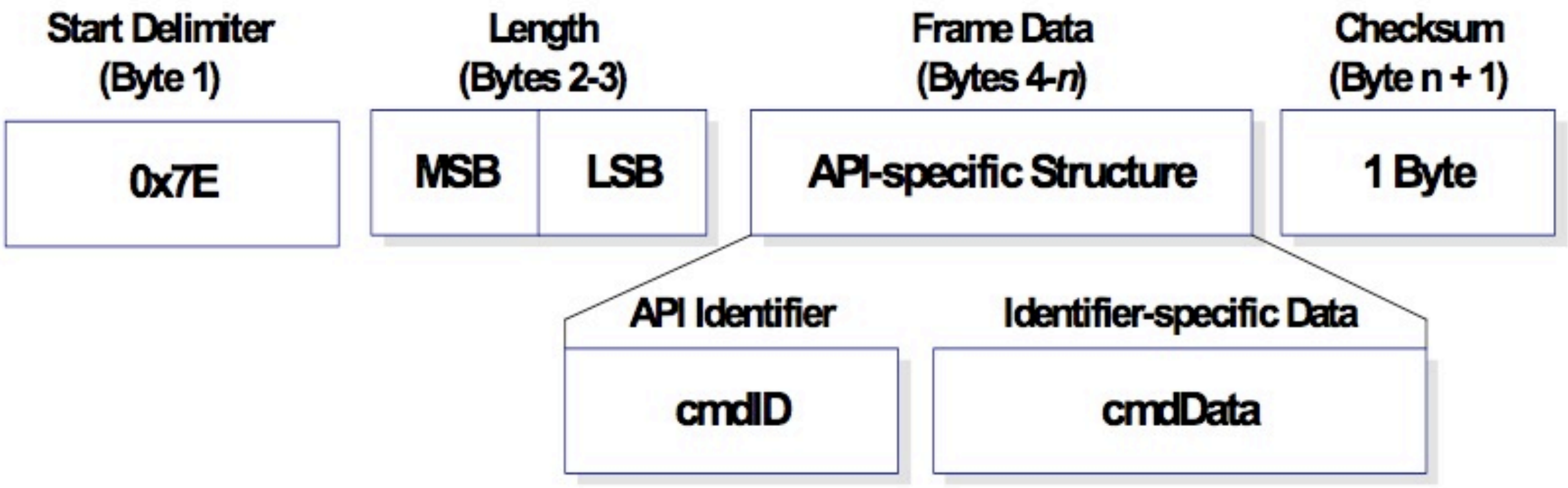
```
sendCommand(ID,0x7777);
```

Envelope Has:

- From address, to address, outside, inside, size, contents, error check



API Basic Frame Envelope



Start Byte

- 0x7E --> also known as the tilde in ASCII: ~

- First thing to do is look for it:

```
  // ARDUINO VERSION:
if (Serial.available() > 0) { // if a byte is waiting in the buffer
    inByte = Serial.read(); // read a byte from the buffer
    if (inByte == 0x7E) {
      // we're at the start of an API frame!
      // add more code here
    }
  }



  // PROCESSING VERSION:
if (port.available() > 0 {
  int inByte = port.read();
    if (inByte == 0x7E) {
      // we're at the start of an API frame!
      // add more code here
}
```

Length Bytes

- MSB: the Most Significant Byte
    - the big part of the number
- LSB: the Least Significant Byte
    - the small part of the number
- bit shift MSB to the right and add it to LSB

```
 // PROCESSING VERSION:
int lengthMSB = port.read(); // high byte for length of packet
int lengthLSB = port.read(); // low byte for length of packet

int lengthTotal = (lengthMSB << 8) + lengthLSB; // bit shift and add for total
```

API Identifier

- Specifies the remaining structure of the frame
  - modem status: 0x8A
  - AT command (immediate): 0x08
  - AT command (queued): 0x09
  - AT command response: 0x88
  - TX request: 0x10
  - TX status response: 0x8B
  - RX packet: 0x90
  - RX packet I/O data: 0x92

```
 // PROCESSING VERSION:
int API_ID = port.read(); // API Identifier indicates type of packet received
```

Identifier-specific Data

- Structures are different for each API identifier and might include:
  - addressing information (333B)
  - status information (received OK)
  - source information (broadcast packet)
  - unstructured data (“Hello World, this is Rob!”)
  - structured data (typically for I/O packets)

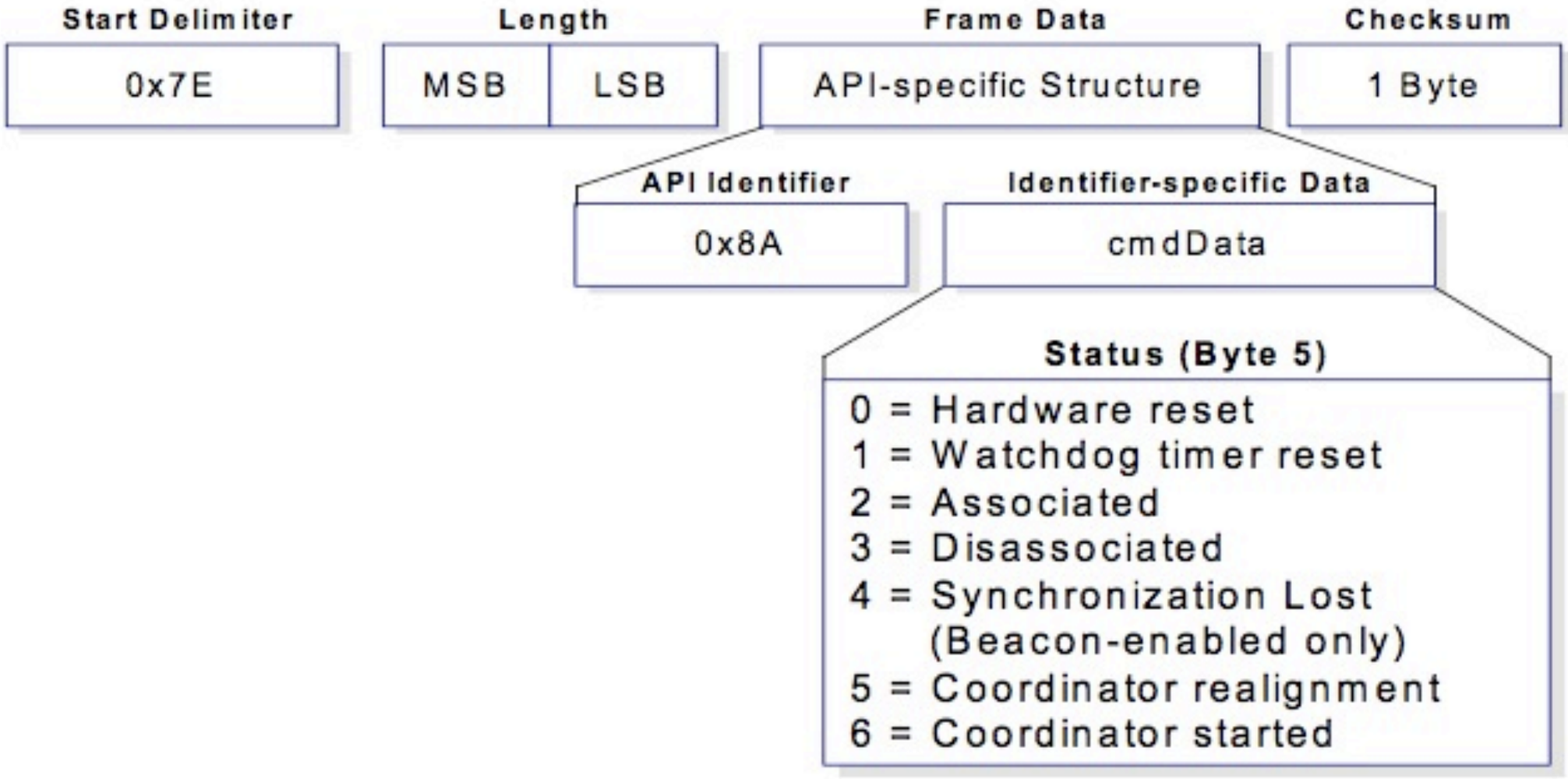
Checksum

- Simple check to detect errors
- To calculate: Not including frame delimiters and length, add all bytes keeping only the lowest 8 bits of the result and subtract from 0xFF.
- To verify: Add all bytes (include checksum, but not the delimiter and length). If the checksum is correct, the sum will equal 0xFF.

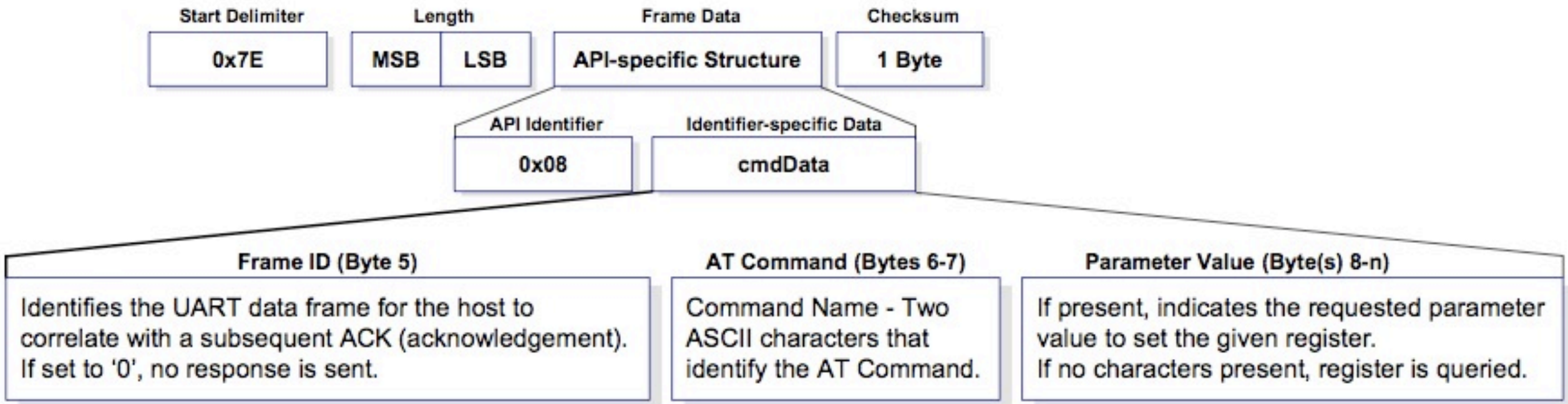
```
 // PROCESSING VERSION:
int localChecksum = (API_ID + addrMSB + addrLSB + RSSI + options + dataSum);

int checksum = port.read();
localChecksum = byte(0xFF -localChecksum);

if ( (byte) checksum - localChecksum == 0) {
  returnVal = dataADC[0];
}
else {
  print("\n\nchecksum error!  " + "\n\n");
}
```

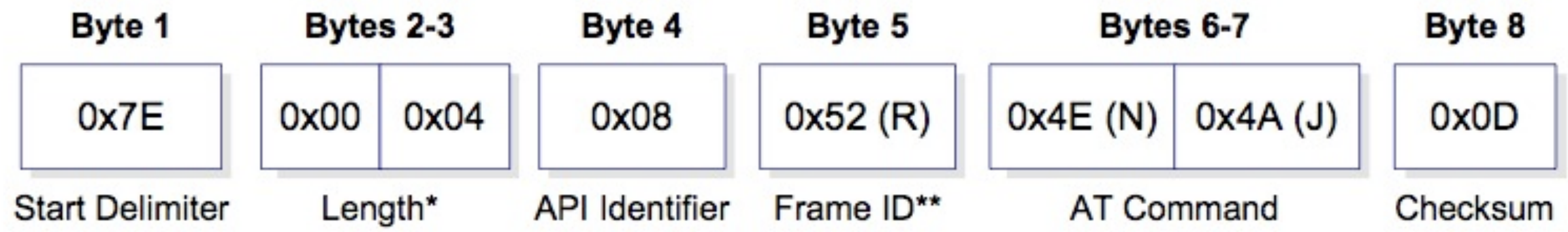
Many Kinds of Envelopes

Modem Status: ZigBee



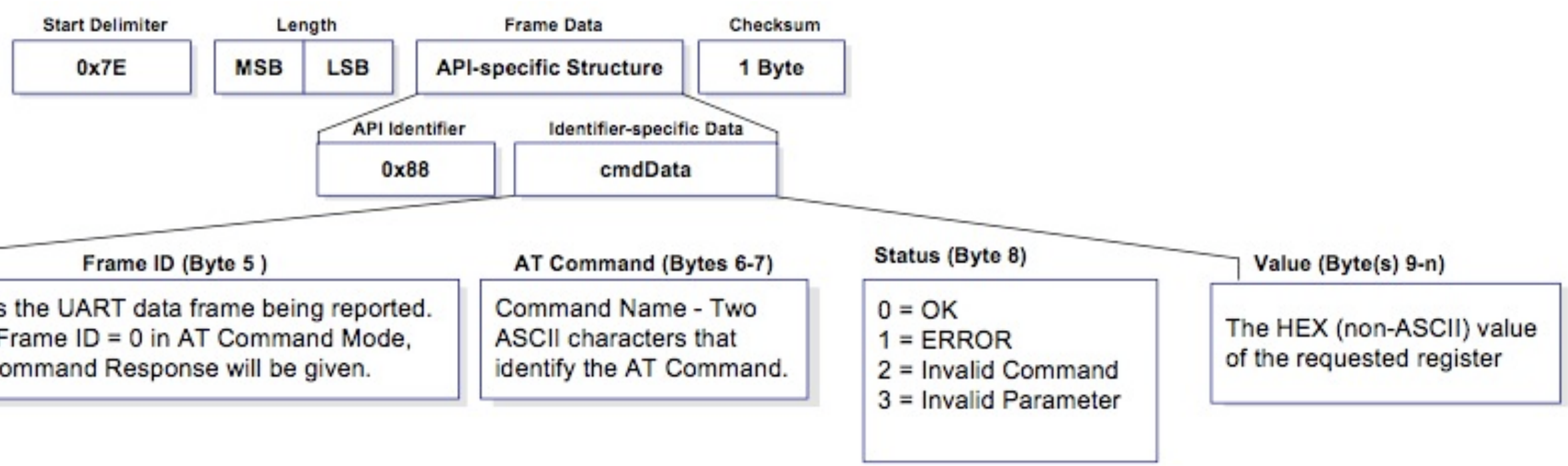
AT Command


Start Delimiter
0x7E
Length
MSB
LSB
Frame Data
API-specific Structure
Checksum
1 Byte
API Identifier
0x08
Identifier-specific Data
cmdData
Frame ID (Byte 5)
Identifies the UART data frame for the host to correlate with a subsequent ACK (acknowledgement). If set to '0', no response is sent.
AT Command (Bytes 6-7)
Command Name - Two ASCII characters that identify the AT Command.
Parameter Value (Byte(s) 8-n)
If present, indicates the requested parameter value to set the given register.
If no characters present, register is queried.


Byte 1
0x7E
Start Delimiter
Bytes 2-3
0x00
0x04
Length*
Byte 4
0x08
API Identifier
Byte 5
0x52 (R)
Frame ID**
Bytes 6-7
0x4E (N)
0x4A (J)
AT Command
Byte 8
0x0D
Checksum

AT Response

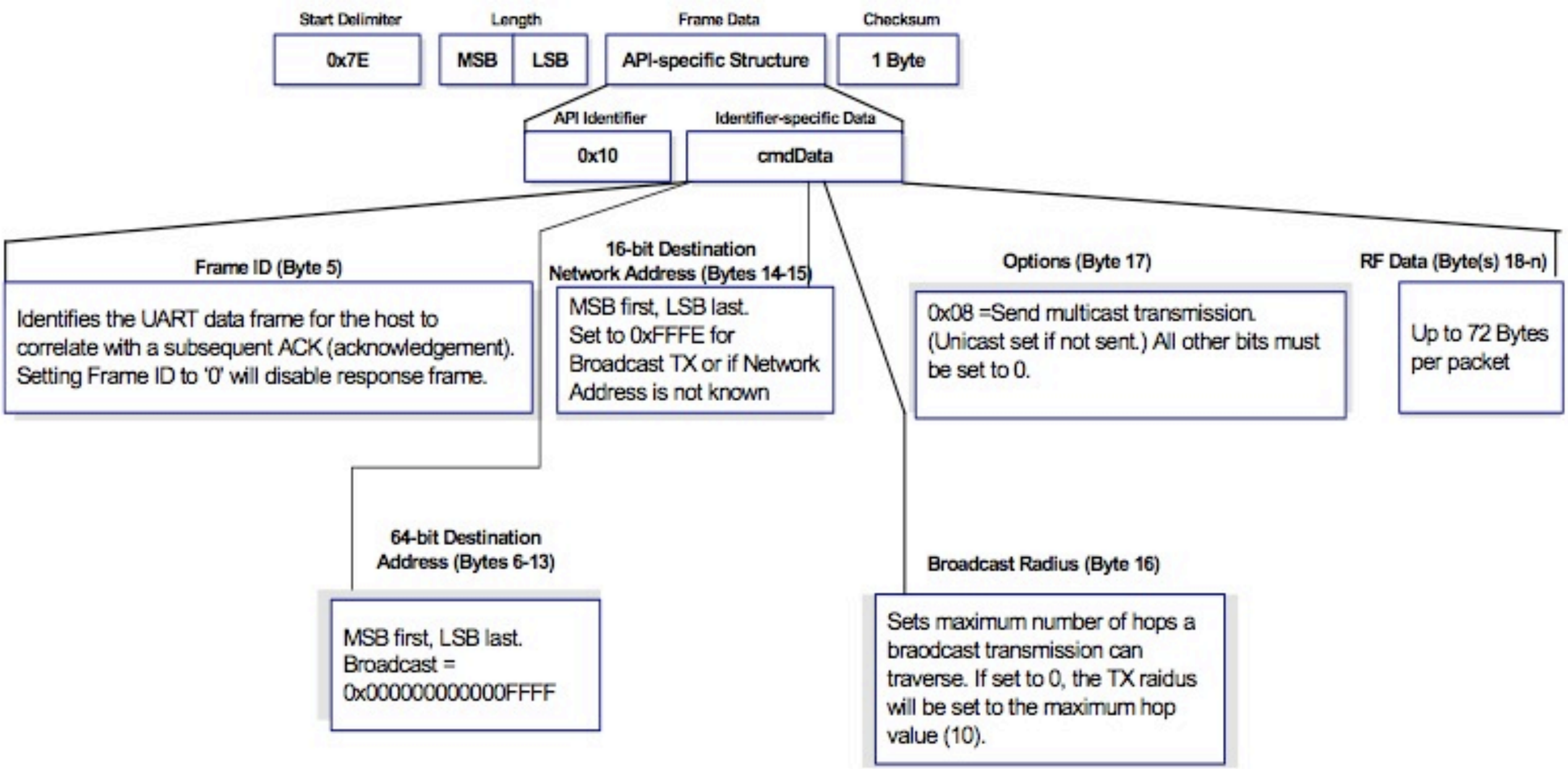
- Frame ID for the response is the same as the matching AT Command request



More API

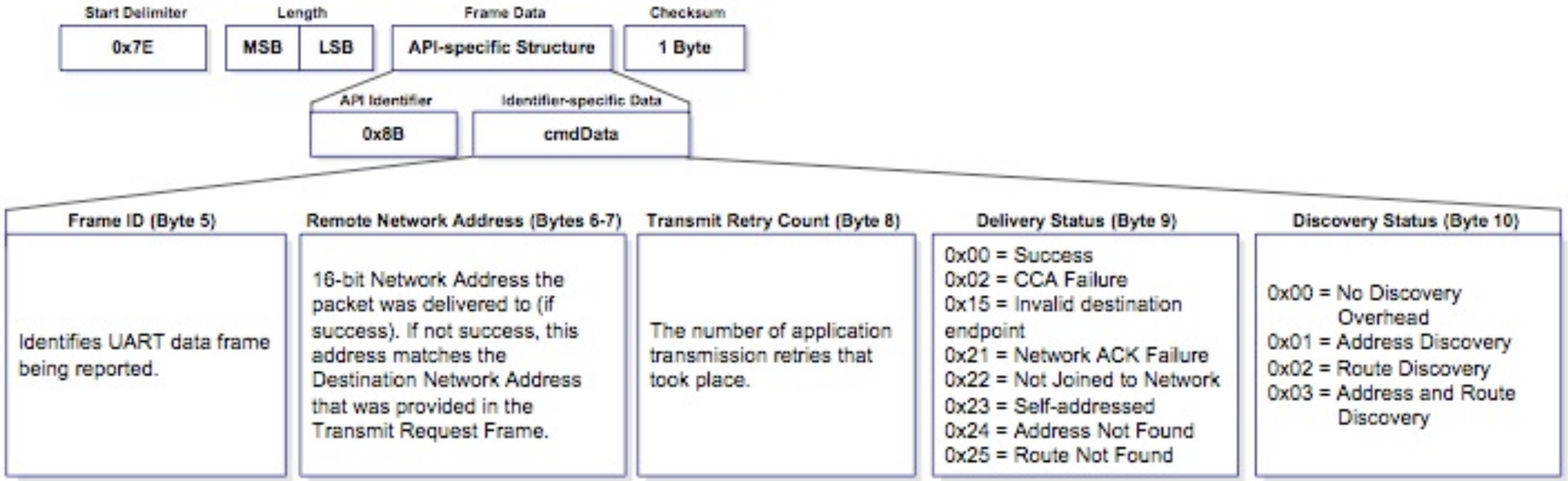
TX (Transmit) Request

- Remember that this is a request. Results can be checked by Frame ID



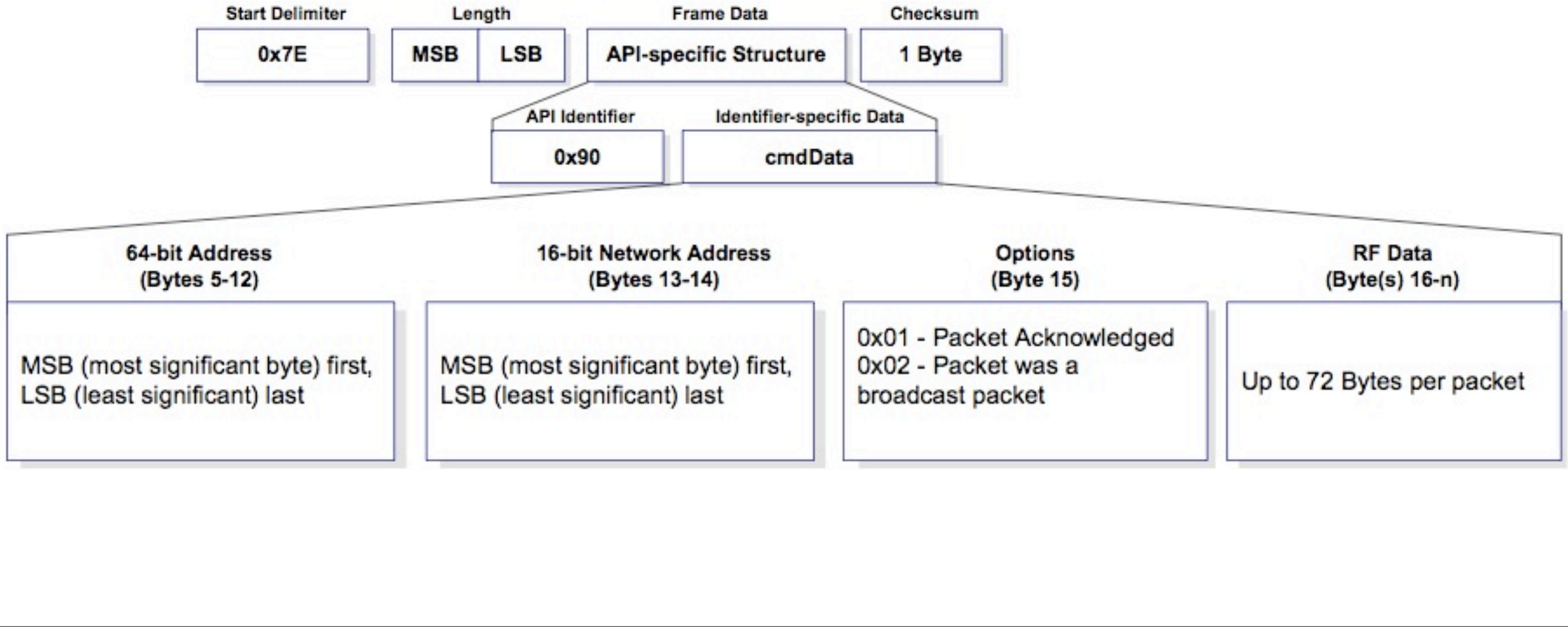
TX Status (Results)

- See if your message was transmitted or not
- Use your Frame ID to see which message is being described

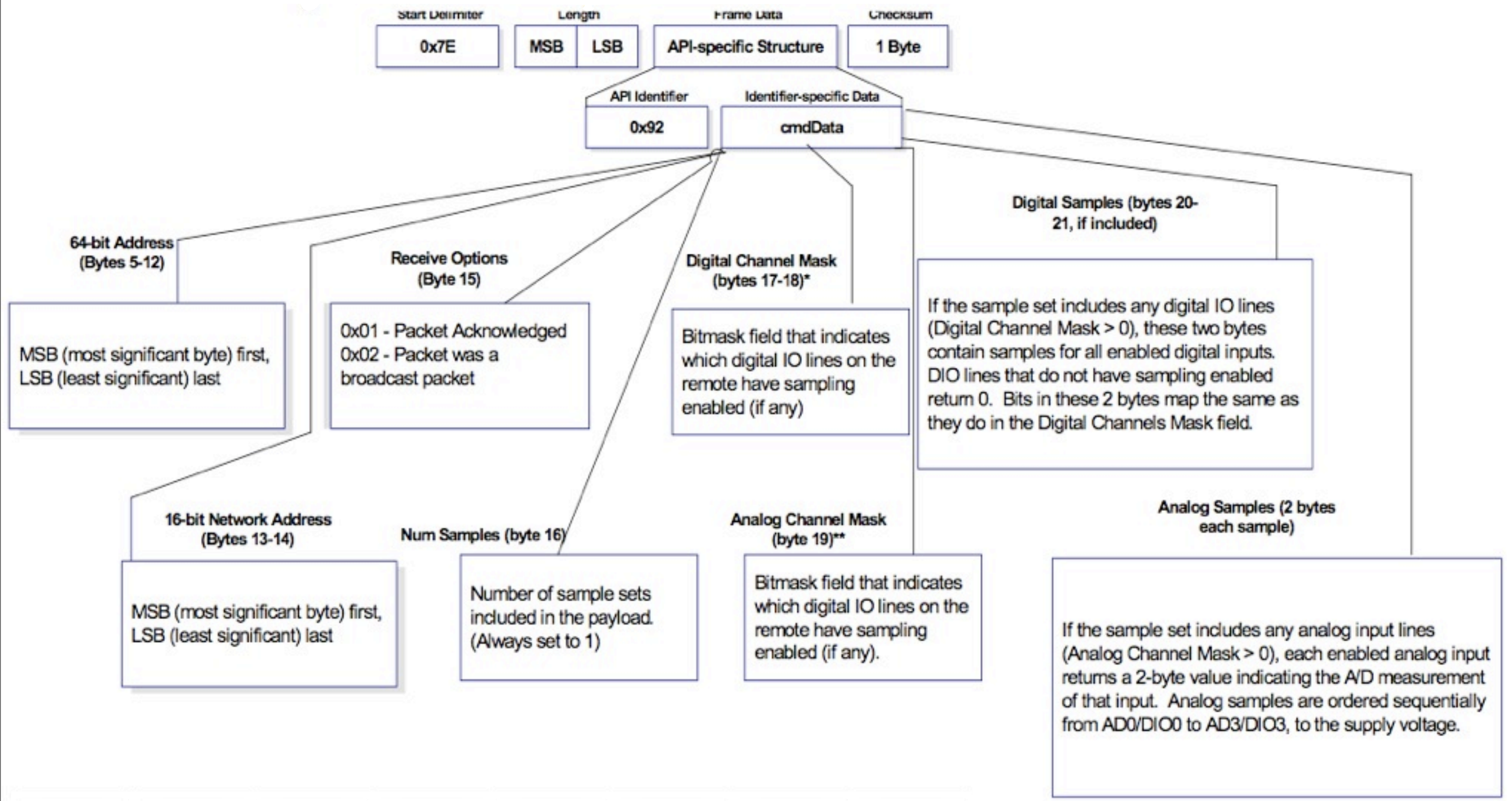


RX Packet

- Maximum of 72 bytes of data per packet
- RF Data section is basis for I/O packets



I/O RX Packet


Start Delimiter
0x7E
Length
MSB
LSB
Frame Data
API-specific Structure
Checksum
1 Byte
API Identifier
0x92
Identifier-specific Data
cmdData
64-bit Address (Bytes 5-12)
MSB (most significant byte) first, LSB (least significant) last
Receive Options (Byte 15)
0x01 - Packet Acknowledged
0x02 - Packet was a broadcast packet
Digital Channel Mask (bytes 17-18)*
Bitmask field that indicates which digital IO lines on the remote have sampling enabled (if any)
Digital Samples (bytes 20-21, if included)
If the sample set includes any digital IO lines (Digital Channel Mask > 0), these two bytes contain samples for all enabled digital inputs. DIO lines that do not have sampling enabled return 0. Bits in these 2 bytes map the same as they do in the Digital Channels Mask field.
16-bit Network Address (Bytes 13-14)
MSB (most significant byte) first, LSB (least significant) last
Num Samples (byte 16)
Number of sample sets included in the payload. (Always set to 1)
Analog Channel Mask (byte 19)**
Bitmask field that indicates which digital IO lines on the remote have sampling enabled (if any).
Analog Samples (2 bytes each sample)
If the sample set includes any analog input lines (Analog Channel Mask > 0), each enabled analog input returns a 2-byte value indicating the A/D measurement of that input. Analog samples are ordered sequentially from AD0/DIO0 to AD3/DIO3, to the supply voltage.

I/O Digital Channel Mask and Digital Data

Digital Channel Mask (bytes 17-18)*

Bitmask field that indicates which digital IO lines on the remote have sampling enabled (if any)

*

N/A	N/A	N/A	CD/DIO 12	PWM/DI O11	RSSI/DI O10	N/A	N/A
CTS/DI O7	RTS/DI O6	ASSOC/ DIO5	DIO4	AD3/DI O3	AD2/DI O2	AD1/DI O1	AD0/DI O0

Digital Samples (bytes 20-21, if included)

If the sample set includes any digital IO lines (Digital Channel Mask > 0), these two bytes contain samples for all enabled digital inputs. DIO lines that do not have sampling enabled return 0. Bits in these 2 bytes map the same as they do in the Digital Channels Mask field.

I/O Analog Channel Mask and Analog Samples

Analog Channel Mask (byte 19)**

Bitmask field that indicates which digital IO lines on the remote have sampling enabled (if any).

**

Supply Voltage	N/A	N/A	N/A	AD3	AD2	AD1	AD0

Analog Samples (2 bytes each sample)

If the sample set includes any analog input lines (Analog Channel Mask > 0), each enabled analog input returns a 2-byte value indicating the A/D measurement of that input. Analog samples are ordered sequentially from AD0/DIO0 to AD3/DIO3, to the supply voltage.

I/O Structure Reviewed

- Num Samples (1 byte)
- Digital Channel Mask (2 bytes)
- Analog Channel Mask (1 byte)
- Two bytes of digital data IF ANY DIGITAL CHANNELS ENABLED followed by...
- ...two bytes for EACH analog channel enabled...
  - Q: How many bytes ATD02 ATD12 ATD23?

I/O Bytes Example

0x7E (start byte)
0x00
0x17 (length)
0x92 (API id)
0x00 (64-bit address)
0x13
0x20
0x00
0x43
0x23
0x12
0xEF
0x03 (16-bit address)
0xA4
0x01 (num samples)
0x00 (digital channel masks)
0x00
0x01 (analog channel mask)
0x02 (first analog sample)
0xF8
0x30 (the checksum)

I/O Code: Basic

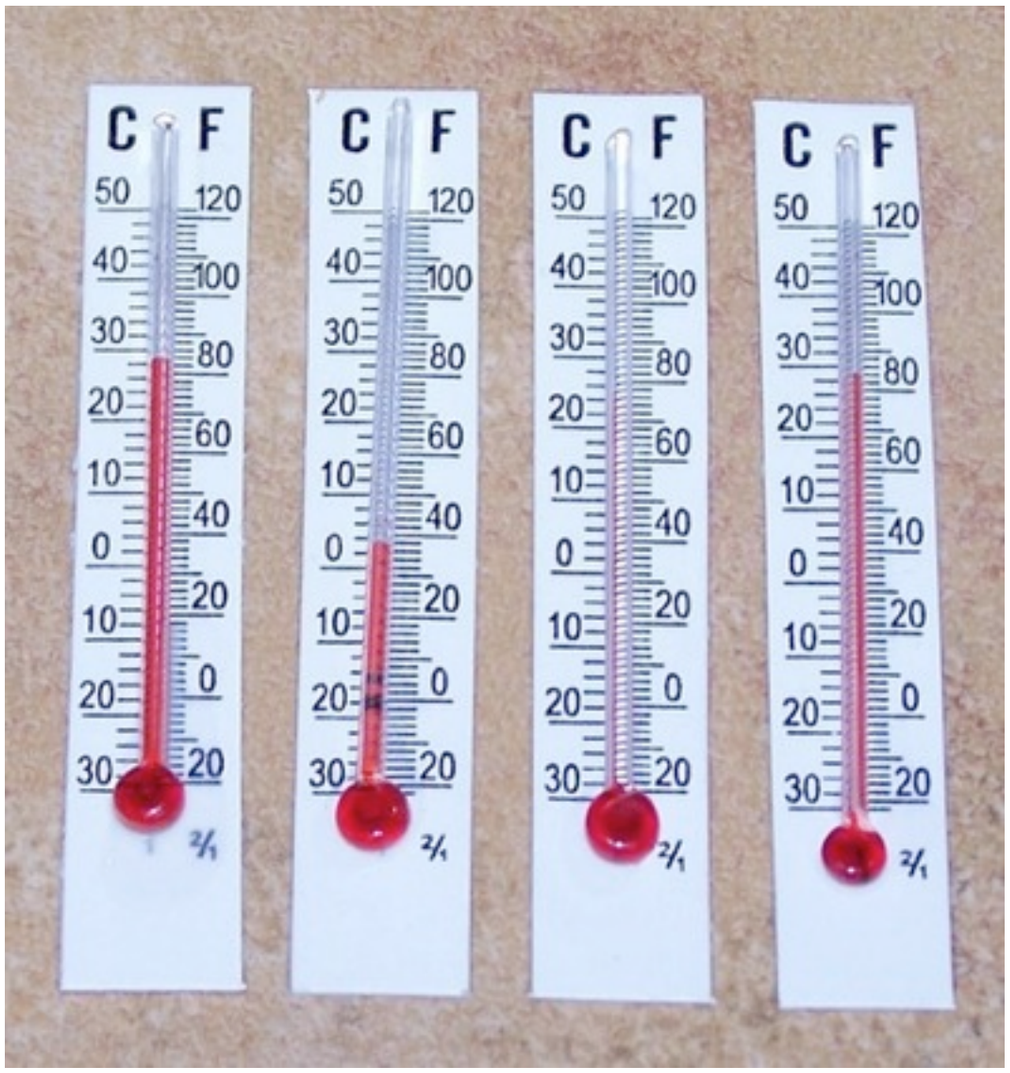
- Fixed parameters make for easier programming

- Assume we are just reading a single ADC channel:

```
  Arduino Version:
// make sure everything we need is in the buffer
  if (Serial.available() >= 21) {
    // look for the start byte
    if (Serial.read() == 0x7E) {
      // read the variables that we're not using out of the buffer
      for (int i = 0; i<18; i++) {
        byte discard = Serial.read();
      }
     int analogHigh = Serial.read();
     int analogLow = Serial.read();
     analogValue =  analogLow + (analogHigh * 256);
    }
  }
```

Simple Sensor Network

API and a Sensor Network



Simple Sensor Network